Google Books API

Main method

*public static void* main(String[] args) *throws* IOException {  
 Scanner scanner = *new Scanner*(System.in);  
 BooksAPI booksService = *new BooksAPI*();  
  
 *while* (*true*) {  
 Map<String, String> params = *getUserInput*(scanner);  
 *if* (params == *null*) {  
 *continue*;  
 }  
 BookOption chosenOption = BookOption.*getByCode*(Integer.*parseInt*(params.*get*("code")));  
 *assert* chosenOption != *null*;  
 params.*remove*("code");  
 System.out.*println*(booksService.*fetchGoogleBookData*(chosenOption.*getOperation*(), params));  
 }  
}  
  
  
*public static* Map<String, String> getUserInput(Scanner scanner) {  
 System.out.*println*("Choose an option:\n" +  
 Arrays.*stream*(BookOption.*values*())  
 .*map*(opt -> opt.*getCode*() + ": " + opt.*getDescription*())  
 .*collect*(Collectors.*joining*("\n")));  
  
 *int* inputCode;  
 *try* {  
 inputCode = Integer.*parseInt*(scanner.*nextLine*());  
 } *catch* (NumberFormatException e) {  
 System.out.*println*("Invalid input. Please enter a number.");  
 *return null*;  
 }  
  
 BookOption chosenOption = BookOption.*getByCode*(inputCode);  
  
 *if* (chosenOption == *null*) {  
 System.out.*println*("Invalid input. Try again.");  
 *return null*;  
 } *else if* (chosenOption == BookOption.EXIT) {  
 System.out.*println*("Exiting...");  
 scanner.*close*();  
 System.*exit*(*0*);  
 }  
  
 Map<String, String> params = IntStream.*range*(*0*, chosenOption.*getPrompts*().length)  
 .*boxed*()  
 .*collect*(Collectors.*toMap*(  
 i -> chosenOption.*getParamKeys*()[i],  
 i -> {  
 System.out.*println*(chosenOption.*getPrompts*()[i]);  
 *return* scanner.*nextLine*();  
 }));  
  
 // Add the chosen code to the map  
 params.*put*("code", String.*valueOf*(inputCode));  
  
 *return* params;  
}

Δημιουργώ έναν HTTP server στη θύρα 8081. Όταν συνδέομαι στο server στη διεύθυνση "/books", τότε ο MyHandler αναλαμβάνει να χειριστεί το αίτημα. Ξεκινώ τον server χρησιμοποιώντας έναν default executor. Μετά δημιουργώ ένα Scanner object για να διαβάσω την είσοδο το user input. Επίσης, φτιάχνω ένα BooksAPI object που μου δίνει πρόσβαση στις άλλες μεθόδους της κλάσης BooksAPI.

Σε ένa while loop, διαβάζω συνεχώς user input και κάνω request μια επιλογή από τις διαθέσιμες που τυπώνονται στο terminal. Αν η είσοδος δεν είναι έγκυρη, ενημερώνω τον χρήστη και επαναλαμβάνεται η ίδια διαδικασία. Επίσης αν η είσοδος είναι invalid, συλλέγω πρόσθετες πληροφορίες από τον χρήστη, ανάλογα με την επιλεγμένη επιλογή που έχει προηγηθεί. Μετά, χρησιμοποιώ το BooksAPI object για να πάρω τα data των βιβλίων από το Google Books API.

# BooksAPI κλάση

Η BooksAPI κλάση κάνει τα requests στο Google Books API. Έχω χρησιμοποιήσει το HttpURLConnection της Java native βιβλιοθήκης Java.net για τα HTTP requests.

Aυτή είναι η βασική μέθοδος

*public* Object fetchGoogleBookData(BookOperation operation, Map<String, String> params) {  
 String apiUrl = *constructUrl*(operation, params);  
 String response = *fetchDataFromGoogleBooksApi*(apiUrl);  
 *return extractDataFromResponse*(operation, response);  
}

και καλεί τρεις μεθόδους που βρίσκονται μέσα στην ίδια κλάση.

Η constructUrl κατασκευάζει και επιστρέφει το URL.

*private* String constructUrl(BookOperation operation, Map<String, String> params) {  
 String apiUrl;  
 *switch* (operation) {  
 *case* SEARCH\_FOR\_BOOKS -> {  
 StringBuilder queryParams = *new StringBuilder*();  
 params.*forEach*((k, v) -> queryParams.*append*("+" + k + ":" + v));  
 String finalQuery = params.*get*("query") + queryParams.*toString*();  
 apiUrl = GOOGLE\_BOOKS\_URL + finalQuery + "&key=" + API\_KEY;  
 }  
 *case* RETRIEVE\_VOLUME\_BY\_ID -> apiUrl = String.*format*(VOLUME\_URL\_TEMPLATE, params.*get*("volumeId"), API\_KEY);  
 *case* FETCH\_PUBLIC\_BOOKSHELVES -> apiUrl = BASE\_URL + USERS + "/" + userId + BOOK\_SELVES;  
 *case* FETCH\_BOOKSHELF\_BY\_ID -> apiUrl = String.*format*(BASE\_URL + "/users/%s/bookshelves/%s/volumes", userId, params.*get*("bookshelfId"));  
 *case* GET\_BOOKSHELF\_VOLUMES -> apiUrl = String.*format*("https://www.googleapis.com/books/v1/users/%s/bookshelves/%s/volumes?key=%s", userId, params.*get*("bookshelfId"), API\_KEY);  
 *default* -> *throw new IllegalArgumentException*("Unsupported operation: " + operation);  
 }  
 *return* apiUrl;  
}

Η fetchDataFromGoogleBooksApi κάνει το HTTP request και επιστρέφει το HTTP response σε String format.

*public* String fetchDataFromGoogleBooksApi(String url) {  
 *try* {  
 String response = *sendHttpRequest*(url);  
 System.out.*println*("Response from API: " + response); // Debugging line  
 *return* response;  
 } *catch* (IOException e) {  
 e.*printStackTrace*();  
 *throw new RuntimeException*(e);  
 }  
}

Και η extractDataFromResponse είναι υπεύθυνη για το deserialization του response.

*private static* Object extractDataFromResponse(BookOperation operation, String response) {  
 ObjectMapper objectMapper = *new ObjectMapper*();  
 objectMapper.*configure*(DeserializationFeature.FAIL\_ON\_UNKNOWN\_PROPERTIES, *false*);  
 *try* {  
 *if* (operation == BookOperation.SEARCH\_FOR\_BOOKS || operation == BookOperation.GET\_BOOKSHELF\_VOLUMES) {  
 VolumeList volumeList = objectMapper.*readValue*(response, VolumeList.*class*);  
 *return* objectMapper.*writerWithDefaultPrettyPrinter*().*writeValueAsString*(volumeList);  
 } *else if* (operation == BookOperation.RETRIEVE\_VOLUME\_BY\_ID) {  
 Volume volume = objectMapper.*readValue*(response, Volume.*class*);  
 *return* objectMapper.*writerWithDefaultPrettyPrinter*().*writeValueAsString*(volume);  
 } *else if* (operation == BookOperation.FETCH\_PUBLIC\_BOOKSHELVES) {  
 BookshelfList bookshelfList = objectMapper.*readValue*(response, BookshelfList.*class*);  
 *return* objectMapper.*writerWithDefaultPrettyPrinter*().*writeValueAsString*(bookshelfList);  
 } *else if* (operation == BookOperation.FETCH\_BOOKSHELF\_BY\_ID) {  
 PublicBookshelfs publicBookshelfContents = objectMapper.*readValue*(response, PublicBookshelfs.*class*);  
 *return* objectMapper.*writerWithDefaultPrettyPrinter*().*writeValueAsString*(publicBookshelfContents);  
 }  
 *try* {  
 *return* objectMapper.*readValue*(response, Map.*class*);  
 } *catch* (IOException e) {  
 *throw new RuntimeException*("Failed to parse API response to a map.", e);  
 }  
 } *catch* (JsonProcessingException e) {  
 *throw new RuntimeException*(e);  
 }  
}

# Enum classes και abstraction

Έχω 2 enum classes, τη BookOperation και τη BookOption. Η ΒοοκΟption είναι ποιο περίπλοκη και χρησιμοποιείται ώστε να αυξηθεί το abstraction στον κώδικα.

Θα αναλύσω πως ο κώδικας αυτός είναι έντονα abstract και άρα scalable με πολύ efficient τρόπο.

1. Αν θέλω να προσθέσω κάποια άλλη επιλογή στο μενού επιλογής του χρήστη, δεν χρειάζεται να αλλάξω τον κώδικα στη main καθόλου. Απλώς προσθέτω ένα καινούργιο entry στη BookOption, πχ

SEARCH\_FOR\_BOOKS(*1*, "Find books", BookOperation.SEARCH\_FOR\_BOOKS, "Query", "query", "Field", "field")

Αυτό σημαίνει ότι ο κώδικας που αληλεπιδρά με τον χρήστη δεν θα χρειαστεί να αλλάξει ξανά ανεξαρτήτως από το πόσες φορές αλλάξουμε, ή αυξήσουμε τα Book operations. Όπως φαίνεται, το τρίτο ΒοοκΟption property είναι ένα BookOperation. Και έτσι γίνεται το γεφύρωμα μεταξύ του BookOption και του ποιου request θα εκτελέσει η BooksAPI κλάση (το instance της).

1. Η getUserInput διαβάζει το input για όλες τις επιλογές του χρήστη, είναι δηλαδή ένα one stop shop solution σχετικα΄με το user input reading.
2. Μετά στη BooksAPI, δεν έχουμε ξεχωριστές μεθόδους για κάθε HTTP requests. Αντί για αυτό, υπάρχει abstraction που σημαίνει ότι όλα τα requests περνάνε από το ίδιο pipeline, το οποίο είναι πολύ efficient. Αν προσθέσω καινούργιο request, πρέπει μόνο να αλλάξω το BookOption enum, προσθέτοντας το κατάλληλο property.

Εν κατακλείδι, η κλάση που διαβάζει το user input kai η κλάση που εκτελεί όλα τα request και το deserialization δεν θα χρειαστούν ξανά άλλαγμα εκτός και αν χρειαστεί αλλαγή η λογική του κώδικα. Για να προσθέσουμε επιπλέον requests, η ΒοοκΟperation και η BookOption είναι τα μοναδικά μέρη όπου πρέπει να αλλάξουμε απλώς προσθέτοντας από ένα καινούργιο entry. Αξίζει να παρατηρήσει κανείς πως η ΒοοκΟption καθορίζει και το user input χρησιμοποιώντας το functionality της γλώσσας για μεταβλητό αριθμό παραμέτρων.

# Data package

Το data package έχει τις κλάσεις υπεύθυνες για τη διαχείριση των δεδομένων που κάνω deserialize από τα HTTP responses, δηλαδή τα data που μου επιστρέφει το Books API της Google. Κατασκεύασα τις κλάσεις αυτές διαβάζοντας το documentation καθώς και με πολύ trial and error.

# Testing

Για testing έφτιαξα διάφορα Servlets ώστε να μπορώ να στείλω CURL requests στο πρόγραμμα ενώ τρέχει. Για παράδειγμα:

*public class* VolumeDetailServlet *extends* HttpServlet {  
  
 *private static final* String GOOGLE\_BOOKS\_ENDPOINT = "https://www.googleapis.com/books/v1/volumes/%s?key=AIzaSyCHj5SSH9cDBUNj\_19AQUhbLS8r1b-B2ZI";  
  
 @Override  
 *protected void* doGet(HttpServletRequest req, HttpServletResponse resp) *throws* ServletException, IOException {  
 String volumeId = req.*getPathInfo*().*substring*(*1*); // This will extract the volumeId from the URL  
 String apiUrl = String.*format*(GOOGLE\_BOOKS\_ENDPOINT, volumeId);  
  
 HttpURLConnection connection = (HttpURLConnection) *new URL*(apiUrl).*openConnection*();  
 connection.*setRequestMethod*("GET");  
 *int* responseCode = connection.*getResponseCode*();  
  
 *if* (responseCode == *200*) {  
 *try* (BufferedReader reader = *new BufferedReader*(*new InputStreamReader*(connection.*getInputStream*()))) {  
 String line;  
 StringBuilder response = *new StringBuilder*();  
 *while* ((line = reader.*readLine*()) != *null*) {  
 response.*append*(line);  
 }  
 resp.*getWriter*().*print*(response.*toString*());  
 }  
 } *else* {  
 resp.*setStatus*(HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR);  
 resp.*getWriter*().*print*("Failed to fetch data from Google Books API");  
 }  
 }  
}

και μετά τα

LocalBookshelfVolumesServlet

και

BookshelfVolumesServlet