Lucene

Celem ćwiczenia jest wykorzystanie biblioteki Lucene do indeksowania i wyszukiwania danych tekstowych z poziomu prostych aplikacji konsolowych Java. Założonym IDE jest IntelliJ IDEA (Community Edition), ale można użyć dowolnego innego IDE wspierającego projekty Maven lub Gradle, ponieważ nie będziemy polegać na kreatorach do tworzenia komponentów aplikacji.

1. Utwórz nowy project wybierając Maven jako system budowania aplikacji.
2. W pliku pom.xml dodaj poniższe zależności. Pierwsza zawiera trzon biblioteki Lucene, a druga jest niezbędna do wykonywania tekstowych zapytań.

```
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-core</artifactId>
            <version>9.9.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-queryparser</artifactId>
            <version>9.9.1</version>
        </dependency>
```

   Zawsze po dodaniu zależności do pliku pom.xml przeładuj projekt (wybierając z menu kontekstowego opcję Maven->Reload lub klikając ikonę Load Maven changes w edytorze).
3. Utwórz w projekcie klasę app.lucene.Main, a w niej metodę main() o standardowej sygnaturze.
4. Umieść w klasie poniższą pomocniczą metodę statyczną, służącą do tworzenia dokumentów w rozumieniu Lucene, reprezentujących książki opisane tytułem i kodem ISBN:

```
    private static Document buildDoc(String title, String isbn) {
        Document doc = new Document();
        doc.add(new TextField("title", title, Field.Store.YES));
        doc.add(new StringField("isbn", isbn, Field.Store.YES));
        return doc;
    }
```

   Dodaj brakujące importy (analogicznie dalej w toku ćwiczenia po dodaniu odwołań do nowych klas/interfejsów).

   Oba pola dokumentu będą indeksowane, ale tylko tytuł ma być wcześniej rozbity na tokeny (pole typu TextField), a kody ISBN będą traktowane całościowo (pole typu StringField). Trzeci argument konstruktorów tworzonych pól decyduje czy wartości pól będą dodatkowo składowane w indeksie w źródłowej postaci. Składowanie zleca się dla pól, które będą wykorzystane jako opis znalezionych dokumentów zwracanych jako wynik zapytania.
5. W metodzie main() dodaj kod tworzący indeks Lucene przechowywany w pamięci i dodaj do niego kilka dokumentów wykonując poniższe kroki:
   a) Utwórz instancję analizatora tekstu. Na początek wykorzystamy StandardAnalyzer, który dzieli tekst na tokeny, zamienia litery na małe oraz rozpoznaje adresy email i URL, ale nie uwzględnia odmiany i nie ma domyślnej stop listy (można ją przekazać poprzez argument konstruktora).

```
    StandardAnalyzer analyzer = new StandardAnalyzer();
```

b) Utwórz obiekt Directory reprezentujący lokalizację, w której będzie przechowywany indeks. W tej chwili wykorzystamy implementację składującą indeks w pamięci na stercie.

```
Directory directory = new ByteBuffersDirectory();
```

c) Utwórz konfigurację dla obiektu IndexWriter powiązaną z utworzonym wcześniej analizatorem tekstu, a następnie sam obiekt IndexWriter bazujący na tej konfiguracji i powiązany z utworzonym obiektem Directory.

```
IndexWriterConfig config = new IndexWriterConfig(analyzer);

IndexWriter w = new IndexWriter(directory, config);
```

d) Dodaj poniższe polecenia dodające kilka dokumentów do indeksu, tworzące je za pomocą utworzonej wcześniej metody buildDoc().

```
w.addDocument(buildDoc("Lucene in Action", "9781473671911"));
w.addDocument(buildDoc("Lucene for Dummies", "9780735219090"));
w.addDocument(buildDoc("Managing Gigabytes", "9781982131739"));
w.addDocument(buildDoc("The Art of Computer Science",
"9781250301695"));
w.addDocument(buildDoc("Dummy and yummy title", "9780525656161"));
```

e) Zamknij obiekt IndexWriter.

```
w.close();
```

Zamknięcie obiektu IndexWriter powoduje zapis segmentu indeksu Lucene w miejscu docelowym. Ewentualne kolejne zapisy do indeksu spowodują utworzenie kolejnego segmentu. Usunięcie dokumentu z indeksu jest oznaczane w specjalnym pliku powiązanym z segmentem. Z kolei modyfikacja dokumentu jest realizowana jako kombinacja usunięcia i wstawienia. Segmenty indeksu można scalić (ang. merge). W ramach scalania dokumenty oznaczone jako usunięte będą fizycznie usuwane.

6. W dalszej części funkcji main() dodaj kod realizujący zapytanie znajdujące wszystkie dokumenty zawarte w indeksie:

a) Zadeklaruj zmienną, w której będziemy umieszczać zapytania i przypisz do niej zapytanie reprezentujące dopasowanie dowolnego ciągu w dokolnym polu dokumentu.

```
String querystr = "*:*";
```

b) Utwórz obiekt Query będący wynikiem parsowania zapytania tekstowego obiektem QueryParser.

```
Query q = new QueryParser("title", analyzer).parse(querystr);
```

Aby wyszukiwanie działało poprawnie, zapytania muszą być parsowane z wykorzystaniem tego samego analizatora tekstu, który był użyty przy indeksowaniu. Został on przekazany jako drugi argument konstruktora parsera. Pierwszy argument wskazuje pole dokumentu, którego domyślnie będą dotyczyć zapytania, w których pole nie zostanie wskazane jawnie. Będziemy z tego korzystać w większości kolejnych zapytań.

c) Wstaw poniższy kod, który wykona zapytanie ograniczając listę wyników do 10 najlepiej pasujących dokumentów. Wykonanie zapytania składa się z kilku instrukcji. Najpierw należy otworzyć indeks uzyskując obiekt IndexReader. Następnie na bazie tego obiektu tworzy się obiekt IndexSearcher. IndexSearcher wyszukuje dokumenty pasujące do podanego zapytania, ograniczając ich liczbę do podanego limitu. Wynikiem wyszukiwania jest obiekt TopDocs, który zawiera tablicę znalezionych dokumentów

w postaci obiektów ScoreDoc oraz obiekt typu TotalHits, który zawiera całkowitą liczbę dokumentów pasujących do zapytania (w naszym kodzie z niego nie korzystamy).

```
        int maxHits = 10;
        IndexReader reader = DirectoryReader.open(directory);
        IndexSearcher searcher = new IndexSearcher(reader);
        TopDocs docs = searcher.search(q, maxHits);
        ScoreDoc[] hits = docs.scoreDocs;
```

d) Dodaj kod wypisujący na konsoli informacje o znalezionych dokumentach.

```
System.out.println("Found " + hits.length + " matching docs.");

StoredFields storedFields = searcher.storedFields();
for(int i=0; i<hits.length; ++i) {
   int docId = hits[i].doc;
   Document d = storedFields.document(docId);
   System.out.println((i + 1) + ". " + d.get("isbn")
                      + "\t" + d.get("title"));
}
```

ScoreDoc jako wynik wyszukiwania nie stanowi dokumentu, a jedynie udostępnia jego numer (nadawany dokumentom kolejno podczas dodawania dokumentów do indeksu) oraz miarę dopasowania do zapytania (score). W celu zbudowania reprezentacji dokumentu do przedstawienia użytkownikowi należy posłużyć się pomocniczym obiektem StoredFields udostępnianym przez IndexSearcher. Oczywiście struktura uzyskanego w ten sposób dokumentu nie musi być identyczna z dokumentem źródłowym, gdyż będzie ona obejmować tylko pola, które zostały oznaczone jako składowane.

e) Zamknij obiekt IndexReader.

```
    reader.close();
```

f) Uruchom aplikację. W wyniku jej działania powinny zostać wypisane na konsoli informacje o wszystkich dokumentach zawartych w indeksie.

7. Sprawdź jakie dokumenty zwrócą zapytania o dokumenty zawierające słowa:
   a) dummy
   b) and
8. W pliku pom.xml dodaj poniższą zależność z analizatorami dla różnych języków (nie ma wśród nich polskiego).

```
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-analysis-common</artifactId>
            <version>9.9.1</version>
        </dependency>
```

9. Zmień analizator na EnglishAnalyzer i ponownie przetestuj zapytania z punktu 7. Dla obu teraz wyniki powinny być inne. Co z tych przykładów można wywnioskować o działaniu EnglishAnalyzer?
10. W pliku pom.xml dodaj poniższą zależność z analizatorem dla języka polskiego wykorzystującym stemmer Stempel (jest też dostępny analizator wykorzystujący Morfologik).

```
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-analysis-stempel</artifactId>
            <version>9.9.1</version>
        </dependency>
```

11. Zmień analizator na PolishAnalyzer, a instrukcje wstawiające dokumenty do indeksu podmień na:

```
w.addDocument(buildDoc("Lucyna w akcji", "9780062316097"));
w.addDocument(buildDoc("Akcje rosną i spadają", "9780385545955"));
w.addDocument(buildDoc("Bo ponieważ", "9781501168007"));
w.addDocument(buildDoc("Naturalnie urodzeni mordercy",
"9780316485616"));
w.addDocument(buildDoc("Druhna rodzi", "9780593301760"));
w.addDocument(buildDoc("Urodzić się na nowo", "9780679777489"));
```

12. Wykonaj następujące zapytania każdorazowo zmieniając wzorzec przypisany do zmiennej querystr i przeanalizuj zwracane wyniki:
    a) Dokumenty, których pole isbn ma wartość „9780062316097”
    b) Dokumenty, których tytuł zawiera słowo „urodzić”
    c) Dokumenty, których tytuł zawiera słowo „rodzić”
    d) Dokumenty, których tytuł zawiera słowo rozpoczynające się od „ro”
    e) Dokumenty, których tytuł zawiera słowo „ponieważ”
    f) Dokumenty, których tytuł zawiera słowa „Lucyna” i „akcja”
    g) Dokumenty, których tytuł zawiera słowo „akcja”, a nie zawiera słowa „Lucyna”
    h) Dokumenty, których tytuł zawiera słowa „naturalnie” i „morderca” w odległości max 2 słów od siebie
    i) Dokumenty, których tytuł zawiera słowa „naturalnie” i „morderca” w odległości max 1 słów od siebie
    j) Dokumenty, których tytuł zawiera słowa „naturalnie” i „morderca” w odległości max 0 słów od siebie
    k) Dokumenty, których tytuł zawiera słowo „naturalne”
    l) Dokumenty, których tytuł zawiera słowo „naturalne” z tolerancją na literówki

Zadanie do samodzielnego wykonania

Wykorzystując kod stworzony do tej pory utwórz 2 programy konsolowe Index.java i Search.java. Pierwszy z nich będzie tworzył indeks i umieszczał w nim kilka przykładowych dokumentów, a drugi wykonywał zapytanie do indeksu i wyświetlał wyniki na konsoli. Oczywiście oba te programy muszą mieć dostęp do tego samego indeksu, więc zamiast implementacji pamięciowej wykorzystamy indeks składowany na dysku.

Wskazówka: Do utworzenia indeksu w systemie plików można wykorzystać konstrukcję:

Directory directory = FSDirectory.open(Paths.get(INDEX_DIRECTORY));

Skorzystanie ze statycznej metody zamiast jawnego wywołania konstruktora sprawi, że automatycznie zostanie wybrana optymalna implementacja uwzględniająca konkretny system operacyjny. Ścieżka przekazana w przykładzie poprzez stałą może być ścieżką względną (względem katalogu uruchomieniowego aplikacji).

Po zweryfikowaniu, że stworzone aplikacje działają poprawnie:

- obejrzyj jakie pliki na dysku zawierają indeks Lucene

- sprawdź co się stanie gdy uruchomisz 2 razy program tworzący indeks (nie usuwając plików indeksu ręcznie z dysku między uruchomieniami), tj. jak zachowa się wyszukiwanie? jak zmieni się zawartość katalogu z plikami indeksu?