Ejercicio 1: Creación de índices

Paquete es.uam.eps.bmi.search.parsing

- Clase **HTMLSimpleParser** implementa **TextParser**:
  - parse: Dado un texto en formato HTML, se pasa todo a minúsculas, se analiza mediante Jsoup y, por último, limpiamos los caracteres no alfanuméricos.
- Clase **HTMLStopwordsParser** extiende **HTMLSimpleParser**:
  - Constructor: Es necesario suministrar un fichero de *stopwords*, con una línea por cada una de ellas.
  - parse: Además de realizar el parseado de su clase padre, reemplaza todas las apariciones de cada *stopword* de la lista creada en el constructor en el texto a analizar.
- Clase **HTMLStemParser** extiende **HTMLSimpleParser**:
  - Constructor: Es necesario saber que tipo de *stemmizador* se usará durante el parseo. Es posible usar cualquier nombre de clase de SnowballStemmer, sin embargo, se definen dos campos de clase **HTMLStemParser.ENGLISH_STEMMER** para un *stemmizador* genérico en lengua inglesa y **HTMLStemParser.PORTER_STEMMER** para el *stemmizador* Porter.
  - parse: Hace el parseado de su clase padre y pasa cada uno de los *tokens* por el *stemmizador*.
- Clase **HTMLAdvancedParser** extiende **HTMLSimpleParser**:
  - Constructor: Es necesario pasarle tanto la ruta al fichero de *stopwords* como el nombre del *stemmizador* a utilizar.
  - parse: Pasa el texto primero por el método parse del padre, luego por el de **HTMLStopwordParser** y finalmente por **HTMLStemParser**.

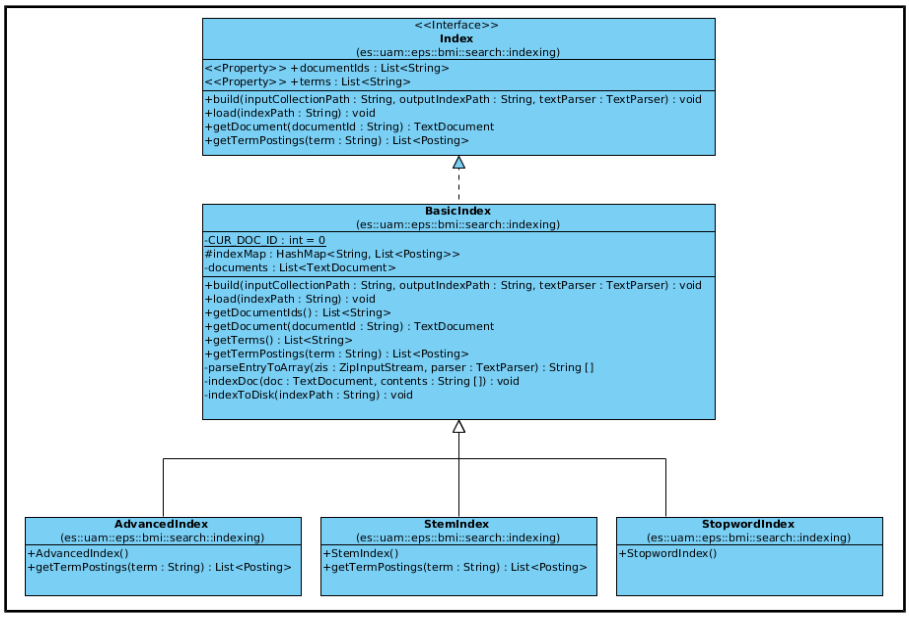
Paquete es.uam.eps.bmi.search.indexing

- Clase **BasicIndex**:
  - build: Dada una colección de documentos comprimidos, parsea cada uno de dichos documentos para obtener un array de términos en formato texto. Con ésta información se instancia un objeto de tipo **TextDocument**, al que se le asigna un identificador de documento para añadirlo a una lista de documentos de la propia clase.
    Por cada uno de estos documentos y su correspondiente contenido, se crean las listas de **Posting** de cada término, añadiendo y actualizando en caso necesario un correspondiente **HashMap** que vincula un **String** término, con una **List<Posting>**.

Una vez leídos, analizados y construido el índice se procede a escribir a disco tanto el índice como la lista de documentos mediante el método **BinIO.storeObject** de la API **fastutil**.

  - load: Mediante el método **BinIO.loadObject** se recuperan tanto el índice como la lista de documentos en memoria.
  - getDocumentIds: Itera sobre toda la lista de **TextDocument** para construir la lista de identificadores en formato **String**.
  - getTerms: Construye una lista del *keyset* del mapa de término-postings.
  - getTermPostings: Realiza un simple **.get(término)** sobre el mapa.

- Clase **StopwordIndex** extiende **BasicIndex**:
  - No se realiza ninguna sobrecarga de métodos de la clase padre, ya que el cambio es el tipo de parseo que se hace, determinado por el **TextParser** pasado al método **BasicIndex#build**.
- Clase **StemIndex** extiende **BasicIndex**:
  - En este caso si se realiza una sobrecarga del método de **BasicIndex#getTermPostings**, ya que es necesario hacer un *stemmizado* del término del que se quieren obtener la lista de **Posting**.
- Clase **AdvancedIndex** extiende **BasicIndex**:
  - Igual que con **StemIndex**, se sobrecarga **BasicIndex#getTermPostings** para *stemmizar* el término pasado como argumento.
- Clase **IndexBuilder**:
  - Contiene un método *main* que construye los 4 tipos de índices arriba descritos dadas las rutas de dicho índice y de la colección de documentos.



Ejercicio 2: Implementación de modelos de recuperación de información

Paquete es.uam.eps.bmi.search.searching

- Clase **BooleanSearcher** implementa **Searcher**
  -