Preface

This master’s thesis has been produced in the period of January 20th through June 30th 2005 at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. The title of the thesis is “Identification of biomedical entities from Medline abstracts using a dictionary-based approach”.

I would like to thank my advisor Heri Ramampiaro.

Trondheim, June 30th, 2005

Magnus Skuland
Abstract

The aim of this paper was to develop a system for identification of biomedical entities, such as protein and gene names, from corpora of Medline abstracts. Another aim was to manage to extract the most relevant terms from the set of identified biomedical terms and make them readily presentable for an end-user.

The developed prototype, named iMasterThesis, uses a dictionary-based approach to the problem. A dictionary, consisting of 21K gene names and 425K protein names, was constructed in an automatic fashion. With the realization of the protein name dictionary as a multi-level tree structure of hash tables, the approach tries to facilitate a more flexible and relaxed matching scheme than previous approaches.

The system was evaluated against a golden standard consisting of 101 expert-annotated Medline abstracts. It is capable of identifying protein and gene names from these abstracts with a 10% recall and 14% precision. It seems clear that for further improvements of the obtained results, the quality of the dictionary needs to be increased, possibly through manual inspection by domain experts. A graphical user interface, presenting an end-user with the most relevant terms identified, has been developed as well.
Contents

1 Introduction 1
  1.1 Problem definition 2 1
  1.2 Thesis outline 2

2 Theoretical foundation 3
  2.1 Molecular biology 3
    2.1.1 Proteins 3
    2.1.2 Amino acids 4
    2.1.3 DNA 4
    2.1.4 Genes 5
  2.2 Bioinformatics 6
  2.3 Information Retrieval 7
    2.3.1 The Boolean model 7
    2.3.2 The vector model 7
    2.3.3 The probabilistic model 8
  2.4 Evaluation 8
    2.4.1 Precision and recall 9
    2.4.2 TREC 10
  2.5 Text Mining 10
  2.6 Biological text mining 12
    2.6.1 Medline 13
    2.6.2 Rule-based approach 14
    2.6.3 Machine-learning approach 15
    2.6.4 Dictionary-based approach 16

3 Previous work 19

4 Own approach 21
  4.1 Basic idea 21
  4.2 The dictionary 22
  4.3 System description 23

5 Implementation 27
  5.1 The prototype application 27
  5.2 The dictionary 28
  5.3 The lookup mechanism 29
  5.4 Postprocessing 32
  5.5 Programming environment 33
    5.5.1 XML 33
List of Figures

1. Amino acid structure ........................................ 4
2. The double helix shape of DNA ............................... 5
3. Rice grains ..................................................... 6
4. Precision and recall ........................................... 9
5. The text mining process ...................................... 11
6. Growth of articles in Medline ................................ 12
7. Entrez databases ............................................. 14
8. Example decision tree ......................................... 16
9. Main view of approach ....................................... 21
10. Class diagram - overview .................................. 24
11. Sequence diagram .......................................... 26
12. Screenshot of the prototype ................................. 28
13. Example hash table-structure at level 0 and 1 .......... 31
14. Steps in postprocessing ..................................... 32
15. Diagram of test results ..................................... 39

List of Tables

1. Processing of the protein dictionary ....................... 29
2. The golden standard .......................................... 37
3. Results from testing .......................................... 38
1 Introduction

The amount of information in biological databases is growing exponentially, reflecting the research effort constantly being performed by teams of biologists and medical researchers world-wide. Medline, the largest biological database available, currently contains close to 15 million articles, growing at a rate of 4% per year. Given these huge amounts of data, researchers require assistance in extracting information and relationships between biomedical entities in this domain. Thus, over the last decade a large research effort has been put into development of various text mining tools that would improve this situation.

Text mining of standard texts, such as news texts, routinely may reach levels of 90-95% on information extraction, but biological text mining is a completely different business. Success rates in this field seldom surpasses 75%. There are many reasons for this phenomenon. For one, biological terms tend to have multiple syntactical variants and synonyms. Scientists in the field do not always stick to established terminologies. Another problem is the lack of data that have been “marked up”, i.e. data indicating the roles played by and the relationship between the entities in the text. A third problem is inconsistencies between databases and the literature. The massive growth of data coupled with inefficiencies in the process of transferring data into other data resources have lead to incomplete databases.

Research in biological text mining have concentrated on three directions: Rule-based approaches, machine-learning approaches and dictionary-based approaches. Additionally, some approaches have been hybrid approaches of the former three. Rule-based systems rely on expert-derived rules, which consist of specific syntactic and semantic properties, to identify biomedical entities in text. Machine-learning systems on the other hand, require the presence of an expert-annotated training corpus. This corpus is then used to automatically derive identification rules through the use of specific statistical algorithms. Dictionary-based systems require the presence of a rich list of protein and gene names to identify occurrences of biomedical entities in text. The identification is often being done by means of various substring matching techniques.
1.1 Problem definition

This paper is focused on the problem of identifying biomedical entities, i.e. gene and protein names, in Medline abstracts. The authors have chosen to attack the problem using a dictionary-based approach. The problem also consists of extracting, from the identified biomedical entities, the most relevant terms to be presented for an end user. The final system should be graphically represented by a prototype interface.

1.2 Thesis outline

Below is a presentation of the contents of this paper.

Section 2 - Theoretical foundation: This section contains knowledge that is important for the understanding of this approach. Fields such as molecular biology, information retrieval and text mining are subjects of focus.

Section 3 - Previous work: This section addresses some of the interesting previous research efforts in the field.

Section 4 - Own approach: This section presents the reader with the core idea of this approach.

Section 5 - Implementation: This section provides information on vital implementational parts. It describes the implemented prototype application and gives insight into the development process.

Section 6 - Results: The results section describes the evaluation of the implemented system and contains information about the system performance.

Section 7 - Discussion: The Discussion section contains an assessment of the approach and its performance. It also contains a final conclusion on the system and outlines directions for further study.
2 Theoretical foundation

This section will provide the reader with an introduction to basic concepts of molecular biology as well as to fields such as bioinformatics, information retrieval and textmining.

2.1 Molecular biology

The study of molecular biology is essentially the study of life. All organisms on the planet are composed of cells, the basic unit of life. Some organisms are single-cell organisms like bacteria, others consist of multiple cells.

Biologists are currently working on problems that affect us in our daily lives every day. They work on finding cures to diseases as cancer and AIDS, and they try to find solutions to the world’s rapidly increasing population. Their accumulation of knowledge is actually life-saving.

In the subsequent sections 2.1.1, 2.1.2, 2.1.3 and 2.1.4, the basic building stones of molecular biology are presented.

2.1.1 Proteins

There are four types of macromolecules that all organisms consist of. These are proteins, nucleic acids, lipids and carbohydrates. These macromolecules are assembled into organelles within the cells.

One can typically divide the functions of proteins into seven categories: Enzyme catalysis, defense, transport, support, motion, regulation and storage.

Enzymes are globular proteins that function as catalysts. They facilitate the breaking up of larger molecules into smaller subunits by stressing certain chemical bonds. Kinases are examples of enzymes. Some proteins have a defense purpose, e.g. spider venom consists of several neural toxins. Transport proteins transport specific ions and small molecules. An example is hemoglobin, that carries oxygen in the blood stream. Support proteins are typically fibers, like for example keratin which forms hair and nails in the human body. Actin and myosin are two motion proteins that facilitate muscle contraction. Regulatory proteins have a key role in controlling many of the body’s functions. Hormones are one type of regulatory proteins. An example is insulin which controls blood glucose levels. Some proteins have a storage
function, in that for example iron can be stored in the cell by binding as ion to specific storage proteins.

2.1.2 Amino acids

Amino acids are the building blocks of proteins. All proteins are simply polymers of 20 different kinds of amino acids, in a specific order.

An amino acid group chemically contains an amino group (-NH\(_2\)), a carboxyl group (-COOH) and a hydrogen atom (H). All of these are bonded to a central carbon atom. See figure 1 for an illustration. The R group indicates the side group that gives each amino acid unique chemical properties.

![Amino acid structure](image)

Figure 1: Amino acid structure

A protein consists of one or more long chains, polypeptides, of amino acids linked by peptide bonds. It was not until the 1950s that scientists, through the work of Sanger, became aware that every kind of protein had a specific amino acid sequence[16].

2.1.3 DNA

DNA\(^1\) was discovered in 1869 by the German chemist Friedrich Miescher. The material he discovered seemed to be located in the nucleus of the cells and also was slightly acidic, hence the name “nucleic acid”.

The three main components in a DNA molecule were determined by P.A. Levene (1920s). They are:

- Five-carbon sugar

\(^1\)Abbreviation for deoxyribonucleic acid.
2.1 Molecular biology

- Phosphate (PO$_4$) group
- Nitrogen-containing (nitrogenous) base

The base may be either a purine (adenine, A or guanine, G) or a pyrimidine (thymine, T or cytosine, C). In humans there is roughly a 30% proportion of both A and T, and a 20% proportion of both G and C. These numbers correspond to Chargaff’s rules, which are given in definition 2.1 below:

**Definition 2.1** The proportion of A always equals that of T, and the proportion of G always equals that of C.

The three main components form what is called a nucleotide, and a single strand of DNA consists of a series of nucleotides joined together in a long strand. Figure 2 shows the double helix shape of a DNA molecule.

![Figure 2: The double helix shape of DNA](image)

DNA is just one of the two types of nucleic acids. RNA$^2$ is the other. Together these two manage the incredible feat of storing and transferring genetic information. RNA, which is similar to DNA in structure, is used to read a cell’s DNA-encoded information and is made as a transcribed copy of portions of the DNA. This transcript ventures out into the rest of the cell, serving as a blueprint specifying a protein’s amino acid sequence. The information stored in DNA can also be passed down to an organism’s descendants, thus the labeling of DNA as “hereditary material”.

2.1.4 Genes

Genes are the sequence of nucleotides that determines the amino acid sequence of a protein. Most genes encode proteins or subunits of proteins, but others produce special forms of RNA.

---

$^2$Abbreviation for ribonucleic acid.
The key of the gene expression was found in 1961 by Francis Crick. He discovered that the genetic code consists of blocks called codons consisting of three consecutive nucleotides. Each codon codes for one specific amino acid. For example GUU and GUC codes for valine, and AAA codes for lysine.

![Rice grains](image)

Figure 3: Rice grains (*Oryza sativa*). Rice actually has a genome consisting of close to 50,000 genes, considerably larger than the human genome.

In 2000, The Human Genome Project reported that they had successfully mapped and sequenced the entire 3.2 billion nucleotide human genome. Interestingly, the human genome consisting of only 30,000 genes plus, is only slightly larger than the genome of mice, and much smaller than that of rice (figure 3).

### 2.2 Bioinformatics

Bioinformatics can be defined as in definition 2.2 below (Tekaia, [15]):

**Definition 2.2** *The mathematical, statistical and computing methods that aim to solve biological problems using DNA and amino acid sequences and related information.*

Thus, one can interpret this in a broader sense to imply that bioinformatics is a field that describes any use of computers to process biological information.

There are multiple fields closely related to bioinformatics. Among these are computational biology (field closer to evolutionary biology than cell and molecular biomedicine), medical informatics (discipline that is concerned with structures and algorithms for the manipulation of medical data), genomics (the science of exploring a species complete genome) and proteomics (studies of gene expression at the level of the functional proteins).
2.3 Information Retrieval

Information retrieval deals with the representation, storage, organization of, and access to information items[1]. It is important to differentiate between the two terms “information retrieval” and “data retrieval”.

Given a collection of documents, a data retrieval merely determines which documents contain the keywords in a user query. This might be unsatisfactory to the user. A relational database system is an example of a data retrieval system.

Information retrieval on the other hand aims at retrieving information about a subject from mostly natural language text, whereas data retrieval systems works on data with well defined structure. The IR system has to extract syntactic and semantic information from the text and rank the documents according to relevance compared to the user query. The goal of such a system is to return the relevant documents.

There exists three classic IR models; the Boolean, the vector and the probabilistic models, and these will be briefly presented here.

2.3.1 The Boolean model

This model is a simple one based on set theory and Boolean algebra. The queries are specified as Boolean expressions. In this model the index terms are considered to be either present or absent in a document. Therefore the index term weights are assumed to be binary (0 or 1), thus leading to a prediction that a document is either relevant or non-relevant. This exact matching may lead to retrieval of too many or too few documents.

2.3.2 The vector model

The vector model differs from the Boolean model in that index terms can be given non-binary weights. This makes partial matching possible. Both the index terms in queries and in the documents are weighted. These can be represented by a query vector and a document vector. In this model it is possible to calculate the degree of similarity between documents, with regard to the query, as the correlation between the two vectors. This can be quantified by the cosine of the angle between them. Thus the vector model proposes a ranking of the document according to the degree of similarity to the query. The value of similarity is a number between 0 and +1.
In the vector model, the index term weights are often quantified using the *tf-idf-term-weighting scheme*. This is a model based on term frequency and inverse document frequency. The tf-idf scheme balances the effects of both intra-cluster similarity and inter-cluster dissimilarity. The normalized frequency \( f_{i,j} \) of a term \( k_i \) in document \( d_j \) is given as in equation 1:

\[
f_{i,j} = \frac{freq_{i,j}}{\max_l freq_{l,j}} \tag{1}
\]

where \( freq_{i,j} \) is the raw frequency of term \( k_i \) in \( d_j \). The maximum is computed over all mentioned terms in \( d_j \). The inverse document frequency is given as in equation 2:

\[
idf_i = \log \frac{N}{n_i} \tag{2}
\]

where \( N \) is the total number of documents in the system and \( n_i \) is the number of documents in which the index term \( k_i \) appears. Given the normalized and inverse document frequencies, it is possible to define the term weight in this scheme as in equation 3:

\[
w_{i,j} = f_{i,j} \cdot \log \frac{N}{n_i} \tag{3}
\]

### 2.3.3 The probabilistic model

The probabilistic model, which was proposed already in 1976, takes as an assumption that there exists an ideal answer set \( R \) which contains exactly the relevant documents in a collection. All index term weights are binary, and the set \( R \) has to be initially guessed. The model assigns to each document a ratio which is the probability that the document is relevant to the query divided by the probability that it is non-relevant. This ratio is then used as a measure of the document’s similarity to the query. Documents are ranked in decreasing order of their probability of being relevant.

### 2.4 Evaluation

To evaluate the performance of a certain information retrieval strategy, it is common to compare the documents retrieved by the strategy with the documents that have been found to be relevant by experts, with regards to similarity. The two most commonly used evaluation measures are presented in 2.4.1. Section 2.4.2 presents TREC, a well known test collection used in evaluation of information retrieval systems.
2.4 Evaluation

2.4.1 Precision and recall

When evaluating the performance of an information retrieval system, the two most frequently used measures are recall and precision. Recall is defined as the fraction of the relevant documents which has been retrieved. Precision is the fraction of the retrieved documents which is relevant [1].

Given a set $R$ of relevant documents and a document answer set $A$, then the intersection of these two sets is identical to the relevant documents in the answer set. Recall and precision can then be described as in equations (4) and (5):

\[
Recall = \frac{|Ra|}{|R|} \quad (4)
\]

\[
Precision = \frac{|Ra|}{|A|} \quad (5)
\]

$|Ra|$ is the number of relevant documents in the answer set. Figure 4 illustrates this. There are however some disadvantages of using these two measures. The estimation of maximum recall for a query requires detailed knowledge of all the documents in a collection. This implies that for large collections, such as biomedical corpora, it is impossible to estimate recall precisely. In some occasions it might be of greater interest to use a single value that combines precision and recall. [2] introduces a measure called Precision-Recall break even point (PRBE), which is the point within ranking where precision equals recall.

![Figure 4: Precision and recall](image-url)
2.4.2 TREC

TREC (Text REtrieval Conference) is a yearly conference started in the early 1990s in Maryland as a response to the lack of robust testbeds and benchmarks in IR. The first conference was held in 1992. Ninety-three groups from 22 countries were attending the TREC 2003. The last TREC was held November 2004\[3\]. Every participant at this conference has to work on the same collection, the TREC collection.

The TREC collection consists of documents, example information requests and a set of relevant documents for each example information request. The size of this collection has been growing considerably. As of 2004 the size was 426 GB\[4\].

The test collections and evaluation software are available online, thus facilitating testing of own retrieval systems at any time. TREC is therefore an user-friendly tool that might help developers test their own contributions in a standardized fashion.

2.5 Text Mining

Text mining is a field covering various techniques that might be applied to textual data in order to retrieve information from text. Thus it is closely related to IR. Text mining is a subterm of the more general term data mining. Data mining can be defined as in definition 2.3 ([5]):

**Definition 2.3** Data mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data. This encompasses a number of different technical approaches, such as clustering, data summarization, learning classification rules, finding dependency networks, analysing changes, and detecting anomalies.

The definition of text mining is identical except the data is of textual type. The expression “previously unknown” means, in this context, either that it is information that not even the writer knows about, or it means that one should rediscover information that the author did encode in the text. The process of text mining, i.e. information extraction from textual data, is a process of several steps. As can be seen from figure 5, the text is initially preprocessed. This preprocessing consists of syntactic and semantic text analysis. Part-of-speech (POS) tagging is used here. This method labels each word with the corresponding part of speech (noun, verb or adjective). The text is also normally parsed, and a parse tree is generated for each sentence.
The second step is text transformation, in which the documents are divided into the single words. These words are then subject to stemming (the words are identified by their roots). In this step there is also removal of stop words.

In the feature selection step the dimensionality is reduced. Irrelevant features are also identified, so as to limit the noise.

In the text mining step the text may be subject to a supervised learning process (classification) or an unsupervised learning process (clustering). In a classification process, the data (observations, measurements) are accompanied by labels indicating the class of observations. These data are then split into training data and test data for model building purposes. Examples of classification techniques are:

- Bayesian classification
- Decision trees
- Neural networks
- Instance-based methods

In the case of a clustering process, class labels of training data are unknown. The system is then given a set of data with the aim of identifying classes or clusters in the data. The idea is that documents in one cluster are more similar to one another. Repetition of the process will result in more correct sets of documents.
2.6 Biological text mining

Research in the field of text mining these days is especially directed towards biological texts and databases. These databases offer information about genes, gene products, protein structure, metabolic pathways, diseases, organisms, DNA-sequences and so on.

One of the main reasons this field is of great interest is the enormous growth of size in the literature. As can be seen from figure 6, the number of articles in Medline is presently close to 15 million, and this is growing approximately 4% per year\[6\]. Medline will be further presented in section 2.6.1.

![Figure 6: Growth of articles in Medline\[6\].](image)

Text mining is a tough business. To this date it has yet not been developed tools that can analyse more than 30% of English sentences correctly and transform them into a structured formal representation\[6\]. The problem is that one always runs into Zipf's law, i.e. that frequent terms account for a large portion of the text, but a large fraction of the terms appear at low frequency and often only once. Examples of these frequent terms are *the*, *of*, *and*, *in*, *to* and also biological terms like *cells*, *protein* etc.

The absolute standard for any biological text mining tool is the data returned by a curator. A curator is a trained domain-expert who is able to sort out any ambiguities as well as handle the high variability of language described by Zipf's law. No computer-based system can match this at present.

A major problem in identification of biological entities, is the multiple variants of terms. All variants of a term should be considered, these can be
2.6 Biological text mining

synonyms and syntactical variants. There is also a problem with ambiguity, in that for example some gene symbols might correspond to disease names or experimental methods. In these situations the context has to be taken into consideration\textsuperscript{3}[7].

Some of the strategies that have been developed with the aim of identifying biological entities are ad hoc rule-based approaches, machine-learning techniques and dictionary-based approaches as well as hybrid approaches that combine different approaches. The first three will be further presented in sections 2.6.2, 2.6.3 and 2.6.4 respectively.

2.6.1 Medline

The huge MEDLINE database today consists of roughly 15 million articles. It is the NLM's\textsuperscript{4} premier bibliographic database covering the fields of medicine, nursing, dentistry, veterinary medicine, the health care system and the preclinical sciences\textsuperscript{10}. Through its Entrez query system it provides the standard “front end” for biomedical literature search\textsuperscript{8}. It contains bibliographic citations and author abstracts from more than 4800 biomedical journals published in 71 countries. MEDLINE began as MEDLARS in 1964 with batch searching. The first online service was presented in 1971. In the early 90s Medline was made available on the Web. The system has become increasingly popular, now with more than 250M hits per year.

PubMedCentral, a product of NLM, was set up to be a barrier-free, biomedical repository that accepts full text and supporting data\textsuperscript{9}. Each item has a unique and persistent PubMedID. Links are provided in this system to sequence information, in addition to full text documents. Entrez is the software that integrates across the various systems. It is a text-based search and retrieval system that is being used at NCBI\textsuperscript{6} for services including PubMed, Nucleotide and Protein Sequences, Protein Structures, Complete Genomes, Taxonomy and others. Figure 7 below gives a high-level view of the Entrez databases. Authors and publishers are able to submit material to Medline through PubMedCentral, given that the material meets certain criteria.

A typical sample record in PubMed Medline contains data tags informing of the PubMed Unique Identifier, date, title, journal, abstract, authors, pub-

\textsuperscript{3}Example: 'EGFR' may have the meaning 'epidermal growth factor receptor' as well as 'estimated glomerular filtration rate'.

\textsuperscript{4}National Library of Medicine

\textsuperscript{5}Entrez PubMed: \url{http://www.ncbi.nlm.nih.gov/entrez/query.fcgi}

\textsuperscript{6}National Center for Biotechnology Information
2.6 Biological text mining

cation type, MeSH terms and source among others. MeSH stands for Medical Subject Headings and is a controlled vocabulary thesaurus. The MeSH terms are descriptors that help arrange the articles in a hierarchical fashion. There are 22568 descriptors in MeSH[11].

2.6.2 Rule-based approach

The automatical tagging of protein names in text is an important first step towards automatic knowledge extraction from biomedical texts. One approach which has been proposed is systems based on hand-written rules. These rules are expert-derived and usually combine surface clues with word syntactic and semantic properties. Such systems require lots of human analysis, but are easy supportable, adjustable and expandable systems. One example of this approach is the Yapex system proposed by Franzén[12].

The Yapex approach consists of several lexical analysis steps. One is finding feature terms, words that describe the function or characteristics of a protein. Such words can indicate the presence of a protein name. Another step is finding core terms, i.e. words that end in -ase or -in for example. To eliminate false hits various filters are used. Examples of such can be pattern matching filters that filter out chemical formulas, arithmetic expressions and amino acid sequences.
Franzén reported results of precision and recall of roughly 83% under sloppy notion of correctness. This notion implies that any token of the hit matches some token of the answer key. Under strict notion the results were closer to 67%[12].

2.6.3 Machine-learning approach

Machine-learning algorithms have also been a proposed solution to the disambiguation of biomedical entities in text. Such an approach aims to fully automate the process of knowledge extraction. They rely on the presence of an expert-annotated training corpus to automatically derive the identification rules by means of various statistical algorithms[14]. GeneWays is an interesting research effort in this direction[13].

This system is an example of unsupervised learning, in that only raw text and no human input or annotation is available to the system. It uses an automated script to download articles that appear in HTML, then it converts these to the XML-format. Term identification is done by lookup over the GenBank database. It only considers terms that are multi-word entries in GenBank, or if single-words, do not appear in the more than 80 000-entry lexicon of common English words used by Brill’s statistical part-of-speech tagger. For machine learning the system utilizes basic features of the terms. Words that appear near a term are considered basic features to which contextual information can be mapped.

There are several ways of adding positional information to the features. A word bag can be used for the words before a term, another word bag for the words after. Each word can also be annotated with its distance from the term.

Hatzivassiloglou[13] considers three learning techniques for construction of a prediction model over the word features. These are naive Bayesian learning, decision trees and inductive rule learning.

The naive Bayesian learning-method is a method that tries to assign to a term occurrence the class c that maximizes the probability of c given the probability of ε (P(c|ε)). ε is the evidence available to the machine learning algorithm for that occurrence.

Decision trees (figure 8) learning is about recursive partitioning of the feature space into areas corresponding to each class label. The C4.5 implementation of decision tree learning (Quinlan, 1993) is well known. The process ends
with a leaf node corresponding to a class label.

![Decision Tree Diagram](image)

**Figure 8: Example decision tree**

Inductive rule learning is a method where rules involving tests on features are iteratively constructed. These rules map a specific combination of features to a class label, and are then applied in a sequential fashion during prediction, so that rules where the system has the highest confidence are applied first.

Hatzivassiloglou reported of results ranging from 74-76% accuracy for the three different machine learning techniques on two-way classification over a 9 million words-corpus. The naive Bayes is significantly faster in training and prediction than the two others though. Regarding pure protein-identification the results for precision and recall were 87 and 82% respectively.

### 2.6.4 Dictionary-based approach

This type of approach utilizes a provided list of protein terms to identify protein occurrences in a text. This is normally done by means of various substring matching techniques. According to [14], the dictionary-based approaches outperform rule-based approaches on accuracy. Their performance is closely related to the quality of the dictionaries, and the creation and maintenance of these may be a non-trivial task. But the other two approaches require huge manual efforts as well.

---

*7Labeling of genes or proteins only.*
Egorov[14] presents an interesting effort named ProtScan. Their system uses a combination of curated and non-curated (“raw”) protein name dictionaries. After applying tokenization and filtering algorithms to the target text (Medline abstracts), they initiate protein identification by loading both the raw and the curated dictionaries into the same hash table. The entry from the curated dictionary always takes precedence. The abstracts are processed one sentence at a time, and qualifying token sequences are searched for the presence of dictionary entries by trying all subsequences from long to short and from left to right. A subsequence is tested by calculating its hash value and then doing a hash table lookup.

The results of ProtScan were very promising. From a corpus of 1000 randomly selected and manually labeled Medline abstracts, it managed a 88% recall and 98% precision[14].
2.6 Biological text mining
3 Previous work

Conventional textmining, i.e. information retrieval and extraction performed on standard texts (like news texts), is a research field that has already come of age. Extraction of person and place names from such texts can routinely reach success rates of 90-95%. But when it comes to textmining of biological texts, such as abstracts from the biomedical database Medline, so-called biological textmining, the success rates are much lower, often around 70%. Biological textmining is currently a field that is subject to huge amounts of research. It is a challenging field, where even something as trivial as a slash-sign may imply two different entities or a single compound.

The three main approaches for identification of biomedical entities in text are ad hoc rule-based approaches, machine-learning approaches and dictionary-based approaches as well as hybrid approaches of the former three.

The rule-based approach requires expert-derived hand-written rules. These systems require lots of human analysis, but are easy supportable and adjustable. The Yapex approach [12], as described in section 2.6.2, is an interesting research effort in this direction. The Yapex system uses multiple lexical analysis steps in searching for patterns in the text. Under strict notion of correctness, this effort reported values for precision and recall of around 67%.

GeneWays is an example of a machine-learning approach. This system facilitates unsupervised learning, in that no human input or annotation is available. The approach considers the usage of three different learning techniques: Naive Bayesian, decision trees and inductive rule learning. The effort reportedly has precision and recall values of 87 and 82% respectively[13]. GeneWays is further described in 2.6.3. Another research effort in this area is that of Wilbur et al.[23]. They analyzed and compared one lexical approach with two statistical machine-learning approaches. The approach that was most successful was one based on a bayesian classifier approach, with a reportedly 97% classification accuracy.

ProtScan[14] is a proposed dictionary approach with astonishingly good results. Their results are supposedly in the range of 88% recall and 98% precision on 1000 randomly selected Medline abstracts. This approach utilizes a combination of curated and non-curated protein name dictionaries. This approach is further described in section 2.6.4.

Textpresso[24] is a somewhat well known ontology-based approach in the world of biological text mining. Textpresso uses an ontology consisting of
33 different classes of biological concepts and a collection of full text of scientific articles split into individual sentences ready to be marked up. Textpresso is organism-specific, currently focusing on the 3800 full text-corpora of *C.elegans*. Values for precision and recall on abstracts are 52% and 45% respectively.

Nenadic[25] presents a terminology-driven approach with very good results. This approach is a hybrid solution combining lexical, syntactical and external similarity between terms. On a corpus of Medline abstracts they reportedly achieved a 99% precision and 74% recall.
4 Own approach

This chapter will provide the reader with an insight into the basic idea of this approach. Succeedingly, the dictionary, which is a core unit in the approach, will be addressed. The user will also be presented with class and sequence diagrams for increased understanding of the approach.

4.1 Basic idea

This approach is based primarily on the idea of identifying biological terms through lookup in a rich dictionary of biomedical names. Figure 9 shows a main view of the approach described in this paper. The quality of the results of this system is in direct correlation with the quality of the dictionary. Hence it is essential that the quality of the dictionary is as high as possible. Our solution is inspired by the work of Egorov et al.[14] and their proposed system; Protscan.

The results of the Protscan system were almost suspiciously good, but this is a topic that will be addressed in section 7. Protscan used a combination of curated and non-curated (“raw”) protein name dictionaries. For construction of the dictionaries they used the LocusLink database and additionally enriched this by incorporating protein names, aliases, gene names etc. from the linked GenBank, GoldenPath and HUGO database entries.

---

The identification of biological entities in the ProtScan solution is done by processing one sentence at a time from the input abstracts. The sentences are scanned for the presence of sequences of tokens from a “protein words” set. These sequences are then scanned for the presence of dictionary entries by trying all of the subsequences from long to short and from left to right. All the dictionary entries are hash table entries, so the look up is done by calculating a potential protein name’s hash value and then scanning through the hash table entries.

This approach differs from the Protscan-solution in that we try to facilitate a more relaxed matching-scheme. We have also implemented the dictionaries as hash tables, but instead of having the names at only one hash table level, we have implemented a tree structure. The tree structure is maximally 10 levels deep.

This approach iterates through every token in the input abstracts, and for each token it tries to match the hash key at the first level in the dictionary. If it is a match, then the consecutive word is tested for a match on the second level of the dictionary and so on. The approach will always try to return the longest possible match. This is an attempt to increase the flexibility since many of the protein names are very long and a 100% match of these seemed unfeasible.

4.2 The dictionary

The dictionary is the Achille’s heel of this approach and of immense importance. There was built one gene name dictionary and one protein name dictionary. The gene name dictionary was developed based on resources from the HUGO Gene Nomenclature Committee\textsuperscript{9}. This dictionary turned out to be about 21K names of size.

The protein name dictionary was developed from UniProt\textsuperscript{10} (Universal Protein Resource). This is the world’s most comprehensive catalog of information on proteins. It is a central repository created by joining the information contained in Swiss-Prot(protein sequence database distributed by European Molecular Biology Laboratory(EMBL)), TrEMBL(contains entries that eventually might be included in Swiss-Prot) and PIR(Protein Information Resource, located at Georgetown University Medical Center). The dictionary eventually became about 425K names of size.

\textsuperscript{9}http://www.gene.ucl.ac.uk/nomenclature/
\textsuperscript{10}http://www.ebi.uniprot.org/index.shtml
4.3 System description

This section gives an overview of the system design of this approach. Figure 10 gives an overview of implemented classes. There are 15 classes in total, but only 13 are executed at run-time. The ProcessDictionary class has only been used during the initial construction of the protein and gene dictionaries, and the Test class was only used during evaluation of system performance. Short descriptions of each class are given below.

Main:
As the name suggest, this class is the main entry point of the application. It starts the GUI window as well as communicating with the Entrez PubMed web resource through an URLConnection. It uses the Parse class for parsing of the output from PubMed. It also starts up the Tokenize class.

GUI:
This class represents the prototype user interface. It invokes the constructor in the Main-class upon a user click. It also starts the BioTermsWindow-class upon another user click.

Parse:
This class extends the DefaultHandler class which is common for classes implementing SAXParsers. It implements a standard Java SAXParser using the SAXParserFactory class. Parse is used to parse XML InputStreams from PubMed and return the results to the Main class.

Tokenize:
The Tokenize class implements logic for separating the individual tokens from the abstract texts. It does not simply remove all signs because that might affect system performance. But it separates all word tokens, and removes signs like e.g. full stop, comma and parentheses. But removing a sign like slash might be unwise, since it is a crucial part of many bio term names. Tokenize also starts up the PreProcessing class.

PreProcessing:
This class is responsible for preparing the sequences of tokens for lookup. It optionally removes purely numerical tokens as well as performing stemming on the tokens. Stopwords are also removed from the set of tokens. The PreProcessing class starts up the LookUp class.

Stopwords:
This class contains logic for removal of stopwords from text. The list of stopwords is based on the stopwords used in Medline.
Figure 10: Class diagram - overview
4.3 System description

Stemmer:
This class can be instantiated every time it is necessary to perform word stemming. The class is an implementation of the Porter stemming algorithm, freely available on the web.

LookUp:
The LookUp class is responsible for initiating the creation of the dictionaries and for the look-up itself. It also starts up the Weight and Result classes.

Weight:
This class performs calculations on the term frequencies. For each term it calculates the normalized frequency and the inverse document frequency. It then proceeds to calculate weights from these frequencies (see section 2.3.2 for the theory). The results are used to determine the relevance of each term in the LookUp class.

Result:
This is a simple class that contains a result ArrayList of bioterms. The results are sent to the GUI.

Dictionary:
The Dictionary class initiates the ProduceHashTables class thus producing all the hash tables. After this initiation Dictionary contains hash tables representing the gene and protein name dictionaries, all ready for lookup processing.

ProduceHashTables:
This class contains logic for creation of both the gene and protein hash tables representing the respective dictionaries.

BioTermsWindow:
If the user wishes to view every single bioterm identified through the lookup process, the BioTermsWindow class will be initiated by a user click in the GUI. The BioTermsWindow class is a simple user interface implemented with the JScrollPane class, thus facilitating scrolling of the list of bioterms.

Figure 11 shows the flow of program control in the system. Program execution starts in Main. All arrows indicate initiation of a class with the object lifelines being the dotted vertical lines.
Figure 11: Sequence diagram. After some time Main initiates Parse and Tokenize. Program control then flows to Tokenize which in turn initiates PreProcessing. PreProcessing initiates Stopwords (and optionally Stemmer) before initiating LookUp. LookUp initiates Dictionary which in turn initiates ProduceHashTable. LookUp then starts up Weight and Result. Finally GUI might initiate BioTermsWindow upon user action.
5 Implementation

This section describes the implemented prototype as well as the general implementation process. Central implementational details are explored at an algorithmical level. Subsequently, some techniques that have been applied to the set of potential biomedical terms in the postprocessing stage are treated. Finally, the programming environment and major technologies are addressed. The source files can be found in appendix B.

5.1 The prototype application

The prototype was developed in Java using in excess of 4K of code lines. Figure 12 shows a screenshot. As the user starts the application from the command line or from a Java Web Start link, only the left window will appear at first. The user is then free to enter a search term in the search text field (the amino acid alanine is used in this example). It is also possible to adjust the settings regarding stemming and removal of purely numerical entities. The user setting in regard of stemming is a boolean that will be sent to the ProduceHashTables class. The application will then proceed to load either stemmed or unstemmed versions of the gene and protein name dictionaries. The setting for removal of numbers is implemented as a boolean that is sent to the PreProcessing class, where action is taken according to its value.

When the user hits enter or the button, the search term will be sent as part of a query to PubMed through its ESearch-service. The 10 first hits in PubMed will be returned as an XML file. This file is parsed, and the PubMed IDs are extracted. These IDs are then used as part of an EFetch call to PubMed, returning the corresponding abstracts as another XML file. This file is parsed, and the abstract texts are extracted. The abstracts are then sent on to Tokenize for further processing.

When the processing of the abstracts are finished the results are presented in the application window. The left column shows the potentially three most relevant terms from the abstracts on a general basis. The column to the right displays the five protein or gene names that the system has identified as the five most relevant. Below these columns, the total number of gene and protein names identified is shown. If the user then wants to look at all identified bio terms, a click at the bottom button will open the right window effectively displaying every single hit from every abstract. All hits are sorted by frequency on an ascending order using QuickSort.
5.2 The dictionary

Since the dictionary is such a vital part of this approach, the time-consuming process of creating it will be discussed here. The 2.6GB XML file, that originally was downloaded from UniProt and parsed\textsuperscript{11}, contained 2326498 entries. This file was far from ready to be used. Several hours had to be spent to improve the quality. Below is table 1 showing several of those steps this file went through. As can be seen from this, less than half of the entries were unique. The steps mentioned in the table consisted of removing names that gave absolutely no meaning. For example “UniRef100 entry” means that we removed entries on the form: “UPI00003A957A UniRef100 entry”. These are just empty entries.

At this point the file had shrunk to a size of 493.814. Later on we discovered that there were areas of improvement in this file. Several of the names were still incapable of matching any names in the abstracts because of their forms. Some names were incredibly long. All names were tested for their length in number of tokens and we found out that the longest “name” was 181 tokens long. There were 12 names longer than 100 tokens, 218 longer than 40 tokens, 5424 longer than 20 tokens and 23,386 longer than 10 tokens. A decision was made to remove all entries longer than 10 tokens. The length

\textsuperscript{11}The parsing and writing to a textfile alone took about 1.5 hrs using a BufferedWriter.
5.3 The lookup mechanism

To realize the flexible approach that was described in section 4.1, we implemented the protein name dictionary as a 10-level deep tree of hash tables. Hash tables are extremely fast for doing lookup, the cost is only $O(1)$ when
you lookup the value of a given key. That is assuming the hash function is an O(1) computation. But the downside is that they require quite some cost for creation and they take up huge amounts of space, this is especially true for this approach. The gene name dictionary on the other hand is 20 times smaller and is also only implemented as a 1-level hash table, hence the costs for its creation are negligible in comparison.

The creation of the hash tables that implement the dictionaries takes place in the ProduceHashTables class. Below is a little piece of pseudocode explaining this:

```java
createProteinHashTables() {
  for all lines in the protein names file
    split the line into a string array s of tokens (max size 10)
    for all tokens in s
      if 0th token:
        if hashtable at level 0 (ht0) already contains the token:
          get the internal hashtable (ha0) which is the value of the token key
          put the token key and internal hashtable to ht0
        else:
          put the token key and a new Hashtable to ht0
      if 1st token:
        if hashtable at level 1 (ht1) already contains the token:
          get the internal hashtable (ha1) which is the value of the token key
          put the token key and ha1 to ht1
          get the internal hashtable at level 0 (ha0)
          put the 0th token and ha0 to ht0
        else:
          put the token key and a new Hashtable to ht1
          get the internal hashtable at level0
          put the token key and a dummy variable to ha0
          put the 0th token and ha0 to ht0
      ...for levels 2 through 9 it is identical as for level 1
```

A hash table consists of (key,value)-pairs. In this implementation we use the protein names as keys, and for each key there is a hash table at the value position containing all the possible keys that this key maps to at the next level (the next token in the name). The number of unique keys at each level is naturally decreasing as there are fewer names of length 8 words than of length 3 words. Below is a list of the number of keys at every level:
5.3 The lookup mechanism

1. 206 304
2. 24 391
3. 33 314
4. 23 086
5. 20 287
6. 9 578
7. 6 212
8. 3 717
9. 2 051
10. 925

Basically, this datastructure becomes very space- and resource-demanding. Our application spends about 24 seconds on building this structure. A huge part of this cost is naturally the I/O-cost of reading the textfile containing the 425K lines of protein names. The application was run on 512 MB of RAM, using the `java -Xmx500m`-command to make sure that the application did not run out of memory. It is not possible to write the hash table-structure to file and read it in again at runtime, this is too memory-demanding. It has to be created and loaded in to memory every time the application is run, but once it is created, lookup is obviously very fast. Figure 13 is an example illustration of the hashtable-structure at level 0 and 1.

![Figure 13: Example hash table-structure at level 0 and 1](image)

The actual lookup itself is done in the readAbstract-method in the LookUp class. This method traverses every token in the abstract text and looks it up
in the gene and protein dictionaries. If it matches a gene, the token is added to the list of identified genes, and the traversal jumps to the next token. If it is a protein match, the lookup will then proceed to try and match the consecutive token in the abstract text with a key in this token’s internal hash table. This will indicate a match of length 2, and the process may follow the pointers to the next level in the hash table-structure, as shown in figure 13, to try and get a match of length 3. The traversal of the abstract text’s tokens will then jump the corresponding number of places, to make sure all tokens are only attempted matched once.

5.4 Postprocessing

The initial result set from the lookup process is rather large. To increase the level of precision, these tokens are subject to postprocessing in the Lookup class. This process consists of several steps. The first step is removing eventual stopwords. Secondly, purely numerical entities are removed. As a third step, entries consisting of 1-2 letters are promptly eliminated. The next step is removing entries that match any entry in a specific postprocessing stopwords-list. This stopwords-list has been developed by studying the 300 most frequent, non-relevant entries that were returned on different input-terms. The fifth step is removal of decimal number-entries. The succeeding step is the removal of entries of length 1 word that do not match any entry in a one-level hash table containing all the protein names in the dictionary. Only entries of length 1 that have a match in this hash table are considered potential protein names. This step dramatically increases precision because of removing entries that initially are assumed to be protein names because of the flexible matching-strategy, but in fact simply are uninteresting first words of longer protein names. The seventh and last step is the removal of entries containing only measures, e.g. \textit{kda}\textsuperscript{12}-entries. Figure 14 summarizes the steps in the postprocessing of the set of potential bioterms.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{postprocessing_diagram.png}
\caption{Steps in postprocessing}
\end{figure}

\textsuperscript{12}Abbreviation for kilodalton.
5.5 Programming environment

Considering it was decided to develop the application in Java and make it available through Java Web Start, a short presentation of these technologies will be given in the consecutive subsections. XML, and parsing of XML documents, which are core technologies in this approach, are also addressed in this section.

5.5.1 XML

XML is a subset of SGML (Standard Generalized Markup Language), designed to ease the exchange of structured documents over the Internet. XML can be dated back to November 14, 1996, the publishing date of the oldest working draft regarding XML currently listed at the W3C, called “Extensible Markup Language (XML)”\(^{18}\). XML consists of logical entities called elements and has clearly defined start and end points. An XML document is extensible and self-describing and can exist on its own without any further definition of its structure\(^{17}\).

Alternatively, an XML document can be defined through a DTD (Document Type Definition) or through XML Schema. Both of these are W3C recommendations\(^{13}\). Drawbacks with DTD is the lack of XML-syntax and its limited expressive power. XML Schema on the other hand is more flexible. It is wellformed XML which allows the user to define datatypes such as string and integer. XML Schema also allows uniqueness constraints and references between elements.

5.5.2 Parsing

Our approach requires handling of XML documents at runtime. To work with XML in programming languages it is necessary to parse the XML documents. There exists two main parsing technologies: DOM and SAX.

The W3C DOM (Document Object Model) is providing a standard set of objects for representing HTML and XML documents and a standard interface for accessing and manipulating them. It is separated into the Core DOM, the XML DOM and the HTML DOM. Only the XML DOM is of interest in this context.

\(^{13}\)This means that they are technologies that have reached a consensus both in- and outside of the World Wide Web Consortium.
The XML DOM is platform and language independent and is a W3C standard. It views XML documents as a tree structure of elements embedded within other elements. All elements can be accessed through the DOM tree. All the nodes in the DOM tree represent objects which have functions and an identity. DOM is very complex and not ideal for parsing large XML documents[19],[20]. JDOM is a simple and Java-specific API that has been developed to exploit special Java features and avoid the complexity of the general DOM API.

SAX (Simple API for XML) is clearly the best choice for parsing of huge XML documents. In SAX an XML tree is not viewed as a data structure, but as a stream of events that is generated by the parser. These events are e.g. encountering an element start tag, an element end tag or character data inside an element. These events invoke a callback method that the programmer writes. A data structure is not necessarily built during parsing, and this is convenient for memory usage reasons. An XML document can be piped through an application very quickly. To use SAX one needs:

- Java 1.1 or higher
- SAX2-compatible XML parser installed on the Java classpath
- The SAX2 distribution installed on the Java classpath

Since we had to parse an XML file of size close to 3 GB to build the protein dictionary, our choice landed on a SAXParser. It was the only choice considering the huge memory cost it actually is parsing such a large file. Using a DOM parser would most likely have required a larger RAM on the system.

5.5.3 Java

Java is an object-oriented programming language that can be traced back to May 1995. At this point Sun Microsystems released the first Java Development Kit (JDK) through the Internet, making it possible for developers world wide to download and start using it[21]. 1998 saw the release of Java 2 SDK, Standard Edition (J2SDK).

Java is tightly knit to the Web and is sometimes called the World Wide Web programming language, but it is a powerful programming system that is used in numerous applications. The Java technology consists of both a programming language and a number of specialized platforms. The programming language Java makes it possible to write programs that run in the browser,
from a desktop, on a server or any user application. Java programs are interpreted by a Java Virtual Machine (JVM) for the “native” operative system. This means that any computer system with a JVM installed, is able to run a Java program regardless of where the application was developed. This unique portability is perhaps the greatest strength of Java.

5.5.4 Java Web Start

Java Web Start is a technology that enables standalone Java software applications to be deployed with a single click over the network. It ensures that the most current version of the application will be deployed, as well as the most correct version of the Java Runtime Environment (JRE). Java Web Start is included in the JRE as part of J2SE.

In the spirit of Java’s portability, Java Web Start works with any Web server and browser. The browser-independent architecture is ensured by the Java Network Launching Protocol & API (JNLP)[22].

To use the Java Web Start, a developer has to create a JAR (Java Archive) file containing all necessary .class-files and other resources. Additionally, a manifest-file has to be put inside the JAR file, containing a pointer to the main class of the application. If a client is to be given full access to the system, one also has to sign the JAR file. Finally, a JNLP file has to be produced. This is a small XML file containing additional information about the application.
6 Results

To evaluate the system performance in terms of precision and recall, it was imperative to have a golden standard available for testing. This golden standard had to be biomedical texts that had been tagged by experts, in particular these texts ought to be Medline abstracts. We managed to find suitable data sets for testing on the web[26]. As part of the project “Proteinhalt i text” (Protein concentration in text), a Swedish research group developed the protein name tagger Yapex (described in section 2.6.2). Through their web site they also provide access to two data sets of tagged Medline abstracts as XML files. These data sets were one reference set consisting of 99 abstracts and a test set of 101 abstracts. The protein names from all 200 abstracts were annotated by domain experts connected to the Yapex project. Table 2 shows the statistics of the golden standard data. As the reader will notice, the average numbers of protein names per abstract for these data sets are quite high.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Size</th>
<th>Number of tags</th>
<th>Avg. tags/abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>yapex_ref_collection</td>
<td>99</td>
<td>1559</td>
<td>15.7</td>
</tr>
<tr>
<td>yapex_test_collection</td>
<td>101</td>
<td>1789</td>
<td>17.7</td>
</tr>
</tbody>
</table>

Table 2: The golden standard

The reference collection of 99 abstracts was used to develop and improve our system. We then later on proceeded to test and evaluate the system using the test collection. The results from testing can be found in table 3.

The table shows results from several of the postprocessing steps leading to the final results. We tested the system using four different settings, according to true/false-values of using stemming on the abstracts and removing purely numerical entities.

The first row of results indicates the percentage of stopwords initially removed from the abstracts. “Number of hits” indicates the number of protein hits that are found in the test sets initially. As can be seen, this number is between 3 to 4 times the number of tags in the golden standard. The next 9 rows provide the number of hits with length more than 1 word that have been found. The system did not find any stopwords amongst the bioterm hits. The next row indicates how many purely numerical entities have been removed from the hits. A zero indicates a rN-setting of true. “1-2 signs” indicates how many hits consisting of a word of length 1 to 2 signs that
<table>
<thead>
<tr>
<th></th>
<th>yapex_ref_collection</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>=uS,−rN</td>
<td>=uS,rN</td>
<td>=uS−rN</td>
<td>=uS,rN</td>
<td>=uS−rN</td>
<td>=uS,rN</td>
<td>=uS−rN</td>
</tr>
<tr>
<td>Percent stopwords</td>
<td>0.374</td>
<td>0.374</td>
<td>0.374</td>
<td>0.374</td>
<td>0.363</td>
<td>0.363</td>
<td>0.363</td>
</tr>
<tr>
<td>Number of hits</td>
<td>6223</td>
<td>6050</td>
<td>5062</td>
<td>4868</td>
<td>6165</td>
<td>6050</td>
<td>4811</td>
</tr>
<tr>
<td>Hit of length: 2</td>
<td>682</td>
<td>642</td>
<td>401</td>
<td>355</td>
<td>621</td>
<td>605</td>
<td>381</td>
</tr>
<tr>
<td>3</td>
<td>184</td>
<td>170</td>
<td>62</td>
<td>50</td>
<td>154</td>
<td>150</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>61</td>
<td>51</td>
<td>13</td>
<td>11</td>
<td>49</td>
<td>46</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>14</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Removed stopw.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Num. entities</td>
<td>152</td>
<td>166</td>
<td>103</td>
<td>108</td>
<td>152</td>
<td>166</td>
<td>103</td>
</tr>
<tr>
<td>1-2 signs</td>
<td>367</td>
<td>367</td>
<td>367</td>
<td>367</td>
<td>367</td>
<td>367</td>
<td>367</td>
</tr>
<tr>
<td>PP stopwords</td>
<td>625</td>
<td>636</td>
<td>320</td>
<td>313</td>
<td>625</td>
<td>636</td>
<td>320</td>
</tr>
<tr>
<td>Decimalnumbers</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Nonrel. 1-word</td>
<td>3909</td>
<td>3923</td>
<td>4014</td>
<td>4014</td>
<td>3909</td>
<td>3923</td>
<td>4014</td>
</tr>
<tr>
<td>kda</td>
<td>17</td>
<td>22</td>
<td>7</td>
<td>7</td>
<td>17</td>
<td>22</td>
<td>7</td>
</tr>
<tr>
<td>New size</td>
<td>1205</td>
<td>1182</td>
<td>1224</td>
<td>1215</td>
<td>1205</td>
<td>1182</td>
<td>1224</td>
</tr>
<tr>
<td>Number of unique</td>
<td>632</td>
<td>615</td>
<td>577</td>
<td>577</td>
<td>632</td>
<td>615</td>
<td>577</td>
</tr>
<tr>
<td>Precision</td>
<td>0.1402</td>
<td>0.1387</td>
<td>0.1422</td>
<td>0.1422</td>
<td>0.1402</td>
<td>0.1387</td>
<td>0.1422</td>
</tr>
<tr>
<td>Recall</td>
<td>0.0964</td>
<td>0.0952</td>
<td>0.0973</td>
<td>0.0973</td>
<td>0.0964</td>
<td>0.0952</td>
<td>0.0973</td>
</tr>
</tbody>
</table>

Table 3: Results from testing. uS and rN are short forms of use stemming and remove numbers.

have been identified and removed. “PP stopwords” informs the reader of hits that matched entries in the postprocessing stopwords-list. The next row indicates identified decimal-number-hits. The “Nonrel. 1-word” row indicates the number of entries that have been removed because of being of length 1 and not matching any entry in the 1-level protein hash table. A few entries are also removed containing only measures (kda). The fourth-to-last row shows the new size of the set of potential protein names, which is the size that is to be used for calculation of precision and recall values. The succeeding row shows the number of unique entries in this set of potential hits. The last two rows presents values for precision and recall.

From the results we can immediately see that the number of initial hits is too high. In the test setting of not using stemming and not removing numerical entities, which is the one that gave best results, the number of hits is almost four times the number of tags in the golden standard. We can also notice that 840 hits (13.6%) were of length 2 or higher. Hence the large majority of returned hits are 1-word entries. A large portion of these entries are removed in the “Nonrel. 1-word” step. Out of the 5325 1-word entries in this test setting result set, 4014 were removed (75.4%). It is also clear that the fifth test setting gave the best results for precision and recall. Removing numbers in the preprocessing of the abstracts affects performance in a slightly negative
way. The usage of stemming gave very poor results in this approach. Figure 15 presents the results for precision and recall for all eight test settings in a graphical form. The maximum performance of this system, quantified by evaluating it on 101 previously untested Medline abstracts, is roughly 10% recall and 14% precision.

![Figure 15: Diagram of test results. The left 8 columns show results for precision and the rightmost 8 show results for recall. The numbers 1-8 correspond to the 8 different test settings shown in table 3.](image)
7 Discussion

In this section the reader will be provided with an assessment of the system. The system’s performance will be assessed on basis of what was the initial idea and intentions of the approach. The section ends with a conclusion based on the discussion and finally outlines directions for future work.

An initial reaction to the system performance is slight disappointment. We had hoped to reach results in the region of 30%. But still, it is interesting and rewarding to analyze the results in an attempt to learn from the approach.

The ProtScan approach, which served as an inspiration in the initial stages of developing this approach, delivered results of 98% precision and 88% recall on a test set of 1000 “randomly selected and hand-tagged” Medline abstracts. What immediately stands out to us is the fact that these 1000 abstracts only contained 1914 protein names, i.e. an average of less than 2 protein names per abstract. In fact, 688 of the abstracts in the test set contained no proteins. This seems suspicious. However, it is unfair to compare the results of our approach to the results of the Protscan approach, so we will not dwell more with this.

The hard facts of this approach’s performance is that it identifies approximately 10% of the relevant terms in the golden standard, and approximately 14% of the terms it identifies are relevant terms. The relevant terms that are identified are absolutely perfect matches.

The intentions of this approach (see section 4.1) was to facilitate a more flexible matching-scheme. The idea was to return the longest possible term that hopefully was a match. This still remains a good idea, and reading from the results in section 6, one can see that the best combination of test settings gave 2 matches on 6-word-entries, 14 matches on 5-word-entries, 49 matches on 4-word-entries, 154 matches on 3-word-entries as well as 621 matches on 2-word-entries. Another test setting even gave match to a 9-word-entry. We are certain that these hits would have been very difficult to obtain with a one-level hashtable-approach. Hence, we would argue that it is a good strategy for seeking to get longest possible matches. The weakness of this approach is the one-word non-relevant hits that are returned. Even after removal of all entries that are not present in the one-level protein hashtable, there are still multiple one-word non-relevant hits. While testing the system on the reference collection, many entries of this type were removed from the dictionary. For example there were unique entries in the protein dictionary

\[14\text{The terms are compared with the equals()-method in Java.}\]
for words like “salt”, “fit” and “area”. Removal of every such entry needs to be done manually, preferably by experts and definitively requires a lot more resources than what was available for this project.

The results also show that the usage of stemming has a negative impact on performance in this approach. This has very obvious reasons. When the linked hash table structure is created in useStemming-mode, potentially wrongly linked names will occur. This is because two different second-level words in a name might potentially have identical word stems, and thus be linked to from the same first-level word. In general, stemming seems to have minimal to negative effect on performance on biomedical texts. As Saetre[27] pointed out, protein names are often rooted in Latin or have acronyms as their name. For example, the porter stem of gas (gastrin) becomes ga, which is wrong and too ambiguous.

The removal of numerical entities from the abstracts also seems to slightly decrease the values for recall and precision. This was intended to be a user-option in the interface, and the idea was that one could remove many irrelevant terms such as year-entries and other uninteresting purely numerical entries at the preprocessing step, thus reducing the size of the abstract texts. But this seems unwise, since numbers are a vital part of many protein names.

In the end, the success of this approach comes down to the quality of the dictionary, and then in particular the protein name dictionary. To increase the value of recall, it is imperative that the expert-tagged protein name actually is present in the dictionary. The dictionary used for this approach is not extensive enough, even though the UniProt-resource allegedly is the world’s most comprehensive catalog of information on proteins. The Protscan-project was in possession of biology-experts that manually processed the dictionary clearing out ambiguities and thus dramatically increasing the quality. Usage of such resources was not feasible in this project, but the authors recognize the advantages of this.

Part of the problem with searching in abstracts as opposed to searching full text articles, is the fact that authors are forced to compress information, actually altering the writing style. This fact is reflected in the Textpresso-approach, where the authors actually reported recall values of 44.6% and 94.7% for searching abstracts and full text respectively. Clearly, comparisons with Textpresso puts this approach in a better light. Recall values in the 40%-range does not seem that out of reach, given the potential of improvement of the dictionaries.
7.1 Conclusion

This approach presented a prototype for identification of biomedical terms in Medline abstracts. The prototype allows a user to search for a biomedical term, before it fetches ten abstracts from the Entrez PubMed interface of Medline. These abstracts are then processed, and the user is provided with the most relevant terms identified from these abstracts. The user is also allowed to view all terms that have been identified.

Evaluation of the approach, on a golden standard of annotated data sets of Medline abstracts, shows that the maximum performance of the system equates to a 10% recall and 14% precision. The usage of stemming seems to have a negative impact on performance, partly because of many protein names being acronyms whose word stems might give directly wrong results, and partly because of the distinct core hash table structure. The removal of numerical entities from the abstracts also affects the performance negatively, but only slightly so.

The results revealed from the evaluation are in direct correlation with the quality of the dictionaries that are being used for lookup. Even though multiple hours have been spent improving these, the quality still seems to be inadequate. Improving the dictionaries, as well as expanding them by adding additional protein name resources, is a topic left for future work.

7.2 Future work

The improvement of the dictionaries seems to be an obvious direction for future work on this system. To obtain as perfect results as possible from such an approach, the dictionaries should be manually processed by domain experts, after automatic processing. Adding multiple new different protein name resources to the existing dictionary is also a possibility that should be considered. The gene name dictionary on the other hand seems to be adequate. But it is seldom that gene names are used in the abstracts, it is almost only protein names. So the focus should be on the protein names.

The approach was originally intended to be a hybrid approach, combining dictionary lookup with certain pattern-based rules for identification of biomedical entities. Because of time limitations, this approach was not realized, but it is definitely an interesting direction of study that could yield improved results.
7.2 Future work
This appendix shows examples of the gene and protein name dictionaries that are being used by the system. Both examples contain 35 consecutive entries.

### A.1 Gene name dictionary

1. lmcd1
2. lmln
3. lmna
4. lmnb1
5. lmnb2
6. lmnl1
7. lmnl2
8. lmo1
9. lmo2
10. lmo3
11. lmo4
12. lmo6
13. lmo7
14. lmod1
15. lmod2
16. lmod3
17. lmpd1
18. lmtk2
19. lmtk3
20. lmx1a
21. lmx1b
22. lnpep
23. lax
24. lmx2
25. lhh1cr2a
26. lhh1ocr1
27. lhh1cr2
28. lhh1ocr1
29. lhh1cr1
30. lhh1cr1
31. lhh3cr2a
32. lor
33. lex
34. lexhd1
35. lex1l

### A.2 Protein name dictionary

1. hydroxyethylthiazole kinase family
2. hydroxyethylthiazole kinase putative
3. hydroxyethylthiazole kinase thim/thk
4. hydroxyethylthiazole kinase-like protein
5. hydroxyethylthiazole kinase
6. hydroxyindole o-methyltransferase
7. hydroxyindole o-methyltransferase ec 2.1.1.4 hiomt acetylserotonin o-methyltransferase asmt
8. hydroxyindole o-methyltransferase
9. hydroxyindole o-methyltransferase isoform a
10. hydroxysourate hydrolase
11. hydroxylacyl-coa dehydrogenase
12. hydroxylamine oxidoreductase
13. hydroxylamine oxidoreductase precursor
14. hydroxylamine oxidoreductase
15. hydroxylamine reductase
16. hydroxylamine reductase 1
17. hydroxylamine reductase 2
18. hydroxylaminobenzene mutase
19. hydroxylaminobenzene mutase
20. hydroxylaminochlorobenzene mutase
21. hydroxylase
22. hydroxyase alpha subunit
23. hydroxylase alpha subunit 2
24 hydroxylase beta subunit
25 hydroxylase beta subunit 2
26 hydroxylase blmf
27 hydroxylase component
28 hydroxylase for synthesis of 2-methylthio-cis-ribozeatin in tRNA
29 hydroxylase large subunit
30 hydroxylase molybdopterin-containing subunit
31 hydroxylase nrnh
32 hydroxylase of the polyketide produced by the pks cluster
33 hydroxylase protein
34 hydroxylase putative
35 hydroxylase small component
B Source code

This appendix presents all the Java source files. There are 15 files in total. They are presented in the following order: BioTermsWindow (page 47), GUI (page 49), Main (page 60), Parse (page 65), PreProcessing (page 81), Stopwords (page 88), Stemmer (page 90), Result (page 96), Weight (page 97), Dictionary (page 104), ProduceHashTables (page 105), Test (page 113), ProcessDictionary (page 118) and LookUp (page 128).

```java
package master;

import java.awt.Color;
import java.util.Vector;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.UIManager;

public class BioTermsWindow {
    private Vector v;
    private int xCoordinate;
    private int yCoordinate;
    private int jframewidth;
    private int jframeheight;

    public BioTermsWindow(String name, Vector v, int xCoordinate, int yCoordinate, int jframewidth, int jframeheight) {
        this.v = v;
        this.xCoordinate = xCoordinate;
        this.yCoordinate = yCoordinate;
        this.jframewidth = jframewidth;
    }

    /*
    * Konstruktoren. Denne får som input navnet som skal
    * brukes i hovedvinduet, en vektor med alle biotermene,
    * samt koordinater som bestemmer hvor på skjermen vinduet
    * vil være.
    */
```

```java
        this.jframeheight = jframeheight;
        createGUI(name);
    }

    /*
     * Denne metoden skaper GUIen med navnet som hentes
     * inn i konstruktoren.
     */
    private void createGUI(String name) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame(name);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(440, 540);
        frame.setLocation(xCoordinate + jframewidth + 1, yCoordinate);
        frame.setResizable(false);

        JPanel contentPane = new JPanel();
        contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.PAGE_AXIS));
        contentPane.setOpaque(true);
        frame.setContentPane(contentPane);

        Color col = new Color(255, 255, 240);
        contentPane.setBackground(col);

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.PAGE_AXIS));
        panel.setBackground(col);
        panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        JList list = new JList(v);
        JScrollPane scrollPane = new JScrollPane(list);

        panel.add(scrollPane);
        contentPane.add(panel);

        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) {
        }
        frame.setVisible(true);
    }
```
package master;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GraphicsConfiguration;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;
import java.util.Vector;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import javax.swing.UIManager;

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

private final int jFrameXCoordinate = 200;
private final int jFrameYCoordinate = 242;

private final int jframewidth = 440;
private final int jframeheight = 540;

private int xCoordinate;
private int yCoordinate;
private int height;
private int width;

private Vector v;

public class GUI implements ActionListener, KeyListener{

private JTextField searchTextField;
private JTextArea geneArea, proteinArea;
private JTextArea first, second, third;
private JTextArea one, two, three, four, five;
private boolean useStemming = false;
private boolean removeNumbers = false;
private boolean readyToInsertText = false;
private boolean enterPressed = false;

private int jframewidth = 440;
private int jframeheight = 540;

private final int jFrameXCoordinate = 200;
private final int jFrameYCoordinate = 242;

private int xCoordinate;
private int yCoordinate;
private int height;
private int width;

private Vector v;
public Main main;
public Dictionary d;

/*
 * Konstruktoren initierer GUIen
 */
public GUI()
{
    createGUI();
}

/*
 * Dette er metoden som setter i gang både Main
 * og BioTermsWindow-klassen ved handling fra brukeren.
 */
public void actionPerformed(ActionEvent e)
{

    /*
    * Dersom noen har trykket på "Good luck!"-knappen
    *(eller enter) og tekstfeltet ikke er tomt
    */
    if( (e.getActionCommand().equals("Good luck!")) && !(searchTextField.getText().equals(null)))
    {
        String searchTerm = searchTextField.getText();
        //d = new Dictionary(useStemming);
        main = new Main(searchTerm, this, useStemming, removeNumbers);
    }
    //dersom brukeren ikke ønsker stemming
    else if(e.getActionCommand().equals("Use stemming"))
    {
        System.out.println("stemming here");
        useStemming = true;
    }
    else if(e.getActionCommand().equals("Remove numbers"))
    {
        System.out.println("Remove numbers!!");
        removeNumbers = true;
    }
    else if((e.getActionCommand().equals("All bioterms")) && readyToInsertText)
    {
        System.out.println("aaaaalllllll bioterms");
        BioTermsWindow btw = new BioTermsWindow("iMasterThesis - All identified bioterms", v, jFrameXCoordinate, jFrameYCoordinate, jframewidth, jframeheight);
    }

    public void keyTyped(KeyEvent ke)
    {
    }

    public void keyPressed(KeyEvent ke)
    {
    }

    public void keyTyped(KeyEvent ke)
/*
Denne metoden gjør at et enter-trykk har samme
 effekt som å trykke på "good luck!"-knappen dersom
 man har fylt inn et søkeord i søkefeltet.
*/

public void keyReleased(KeyEvent ke)
{
    if ( ((ke.getKeyCode()) == KeyEvent.VK_ENTER) && !(searchTextField.getText().equals(null)))
    {
        String searchTerm = searchTextField.getText();
        main = new Main(searchTerm, this, useStemming, removeNumbers);
    }
}

/*
Denne metoden får inn arraylisten resultList som
innholder samtlige gen og protein-treff sortert etter
frekvens oppdågende. Størrelsen på geneSortedList
og protSortedList skal være lik.
*/

public Vector receiveResultList(ArrayList resultList)
{
    v = new Vector();
    int abstractNumberCounter = 0;
    int gene = 0;
    String blank = " ";
    ArrayList geneSortedList = (ArrayList)resultList.get(0);
    ArrayList protSortedList = (ArrayList)resultList.get(1);
    // k skal her være 0,2,4,...,18
    System.out.println("genesortedList.size: "+geneSortedList.size());
    for(int k = 0; k < geneSortedList.size(); k+=2)
    {
        System.out.println(k);
        String abstractString = new String("Abstract number: "+
abstractNumberCounter);
        v.add(abstractString);
        v.add(blank);
        v.add(blank);
        abstractNumberCounter++;
        ArrayList sortedGeneKeys = (ArrayList)geneSortedList.get(k);
        ArrayList sortedGeneFrequencies = (ArrayList)geneSortedList.get(k+1);
        ArrayList sortedProteinKeys = (ArrayList)protSortedList.get(k);
        ArrayList sortedProteinFrequencies = (ArrayList)protSortedList.get(k+1);
        String geneNames = new String("Gene occurences: ");
        v.add(geneNames);
        v.add(blank);
        v.add(blank);
        for(int i = 0; i < sortedGeneKeys.size(); i++)
        {
            String geneName = (String)sortedGeneKeys.get(i);
            gene++;
String geneFrequency = ((Integer)sortedGeneFrequencies.get(i)).toString();
String geneString = new String(geneName + " " + geneFrequency);
v.add(geneString);
}
v.add(blank);

String proteinNames = new String("Protein occurrences: ");
v.add(proteinNames);
v.add(blank);

for(int j = 0; j < sortedProteinKeys.size(); j++)
{
String proteinName = (String)sortedProteinKeys.get(j);
protein++;
String proteinFrequency = ((Integer)sortedProteinFrequencies.get(j)).toString();
String proteinString = new String(proteinName + " " +proteinFrequency);
v.add(proteinString);
}
v.add(blank);

System.out.println("Vector size: " + v.size());
System.out.println("Antall mulige gen: " + gene);
System.out.println("Antall mulige protein: " + protein);
insertHits(gene, protein);
readyToInsertText=true;
}
return v;
}*/
public void createGUI()
{
JFrame.setDefaultLookAndFeelDecorated(true);
JFrame frame = new JFrame("iMasterthesis");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(jframewidth, jframeheight);
frame.setResizable(false);
frame.setLocation(jFrameXCoordinate, jFrameYCoordinate);
JPanel contentPane = new JPanel();
contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.PAGE_AXIS));
contentPane.setOpaque(true);
frame.setContentPane(contentPane);

// get the GraphicsConfiguration
GraphicsConfiguration gcon = contentPane.getGraphicsConfiguration();
Rectangle bounds = gcon.getBounds();
xCoordinate = bounds.x;
yCoordinate = bounds.y;
height = bounds.height;
width = bounds.width;

Color col = new Color(255,255,240);
contentPane.setBackground(col);

//panelLeftUpper
JPanel panelLeftUpper = new JPanel();
panelLeftUpper.setLayout(new BoxLayout(panelLeftUpper, BoxLayout.PAGE_AXIS));
panelLeftUpper.setBackground(col);

JLabel searchTerm = new JLabel("Enter search term:");
searchTextField = new JTextField(25);
searchTextField.addActionListener(this);
searchTextField.addKeyListener(this);
searchTextField.setMaximumSize(new Dimension(300,20));
panelLeftUpper.add(searchTerm);
panelLeftUpper.add(Box.createRigidArea(new Dimension(0,10)));
panelLeftUpper.add(searchTextField);

//panelRightUpper
JPanel panelRightUpper = new JPanel();
panelRightUpper.setLayout(new BoxLayout(panelRightUpper, BoxLayout.PAGE_AXIS));
panelRightUpper.setBackground(col);

//søkeknappen som skal være i JPanel panelRightUpper
JButton searchButton = new JButton("Good luck!");
searchButton.addActionListener(this);
searchButton.addKeyListener(this);
panelRightUpper.add(searchButton);

//JPanel som er øverst, layout fra venstre til høyre
JPanel panelUpper = new JPanel();
panelUpper.setLayout(new BoxLayout(panelUpper, BoxLayout.X_AXIS));
panelUpper.setBackground(col);

ButtonGroup bg = new ButtonGroup();
JRadioButton stemming = new JRadioButton("Use stemming");
stemming.setActionCommand("Use stemming");
stemming.addActionListener(this);
stemming.setBackground(col);
JRadioButton nostemming = new JRadioButton("No stemming");
nostemming.setActionCommand("No stemming");
nostemming.setSelected(true);
nostemming.addActionListener(this);
nostemming.setBackground(col);

bg.add(stemming);
bg.add(nostemming);

ButtonGroup bg2 = new ButtonGroup();
JRadioButton removeNumbers = new JRadioButton("Remove numbers");
removeNumbers.setActionCommand("Remove numbers");
removeNumbers.addActionListener(this);
removeNumbers.setBackground(col);
JRadioButton notRemoveNumbers = new JRadioButton("Don't remove numbers");
notRemoveNumbers.setActionCommand("Don't remove numbers");
notRemoveNumbers.setSelected(true);
notRemoveNumbers.addActionListener(this);
notRemoveNumbers.setBackground(col);

bg2.add(removeNumbers);
bg2.add(notRemoveNumbers);

JPanel stemmingPanel = new JPanel();
stemmingPanel.setLayout(new BoxLayout(stemmingPanel, BoxLayout.PAGE_AXIS));
stemmingPanel.setBackground(col);
JPanel numbersPanel = new JPanel();
numbersPanel.setLayout(new BoxLayout(numbersPanel, BoxLayout.PAGE_AXIS));
numbersPanel.setBackground(col);

stemmingPanel.add(stemming);
stemmingPanel.add(Box.createRigidArea(new Dimension(0,1)));
stemmingPanel.add(nostemming);
stemmingPanel.setBackground(col);
stemmingPanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(Color.lightGray),stemmingPanel.getBorder()));

numbersPanel.add(removeNumbers);
numbersPanel.add(Box.createRigidArea(new Dimension(0,1)));
numbersPanel.add(notRemoveNumbers);
numbersPanel.setBackgroud(col);
numbersPanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(Color.lightGray),numbersPanel.getBorder()));

radioButtonPanel.add(stemmingPanel);
radioButtonPanel.add(Box.createRigidArea(new Dimension(0,10)));
radioButtonPanel.add(numbersPanel);
radioButtonPanel.setBackground(col);

JPanel radioButtonPanelContainer = new JPanel();
radiobuttonPanelContainer.setLayout(new BoxLayout(radiobuttonPanelContainer, BoxLayout.LINE_AXIS));
radiobuttonPanelContainer.add(radioButtonPanel);
radiobuttonPanelContainer.setBackground(col);

//JPanel for de tre beste treff
JPanel topthreePanel = new JPanel();
topthreePanel.setLayout(new BoxLayout(topthreePanel, BoxLayout.PAGE_AXIS));
topthreePanel.setBackgroud(col);

JLabel top3 = new JLabel("Top 3 terms:");

//1. mest relevante term
JLabel fir = new JLabel("1:");
first = new JTextArea(1,10);
354  first.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
355  createLineBorder(Color.cyan), first.getBorder()));
356  first.setEditable(false);
357  first.setMaximumSize(new Dimension(300, 60));
358
359  //2. mest relevante
360  JLabel sec = new JLabel("2:");
361  second = new JTextArea(1, 10);
362  second.setMaximumSize(new Dimension(300, 60));
363  second.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
364  createLineBorder(Color.cyan), second.getBorder()));
365  second.setEditable(false);
366
367  //3. mest relevante
368  JLabel thir = new JLabel("3:");
369  third = new JTextArea(1, 10);
370  third.setMaximumSize(new Dimension(300, 60));
371  third.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
372  createLineBorder(Color.cyan), third.getBorder()));
373  third.setEditable(false);
374  third.setMaximumSize(new Dimension(300, 60));
375
376  toptthreePanel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
377  toptthreePanel.add(top3);
378  toptthreePanel.add(Box.createRigidArea(new Dimension(0, 10)));
379  toptthreePanel.add(fir);
380  toptthreePanel.add(first);
381  toptthreePanel.add(Box.createVerticalGlue());
382  toptthreePanel.add(sec);
383  toptthreePanel.add(second);
384  toptthreePanel.add(Box.createVerticalGlue());
385  toptthreePanel.add(thir);
386  toptthreePanel.add(third);
387  toptthreePanel.add(Box.createVerticalGlue());
388  toptthreePanel.setMinimumSize(new Dimension(170, 208));
389  toptthreePanel.setMaximumSize(new Dimension(170, 208));
390  toptthreePanel.setSize(new Dimension(170, 208));
391
392  //JPanel for de fem beste biotreff
393  JPanel topfivePanel = new JPanel();
394  topfivePanel.setLayout(new BoxLayout(topfivePanel, BoxLayout.PAGE_AXIS));
395  topfivePanel.setBackground(col);
396
397  JLabel top5bio = new JLabel("Top 5 bioterms:");
398  JLabel oneLabel = new JLabel("1:");
399  one = new JTextArea(1, 5);
400  one.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
401  createLineBorder(Color.cyan), one.getBorder()));
402  one.setEditable(false);
403  one.setMaximumSize(new Dimension(300, 60));
404
405  JLabel twoLabel = new JLabel("2:");
406  two = new JTextArea(1, 10);
407  two.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
408  createLineBorder(Color.cyan), two.getBorder()));
409  two.setEditable(false);
410  two.setMaximumSize(new Dimension(300, 60));
411  JLabel threeLabel = new JLabel("3:");
three = new JTextArea(1,10);
three.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
    createLineBorder(Color.cyan), three.getBorder()));
three.setEditable(false);
three.setMaximumSize(new Dimension(300,60));

JLabel fourLabel = new JLabel("4:");
four = new JTextArea(1,10);
four.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
    createLineBorder(Color.cyan), four.getBorder()));
four.setEditable(false);
four.setMaximumSize(new Dimension(300,60));

JLabel fiveLabel = new JLabel("5:");
five = new JTextArea(1,10);
five.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
    createLineBorder(Color.cyan), five.getBorder()));
five.setEditable(false);
five.setMaximumSize(new Dimension(300,60));

topfivePanel.add(top5bio);
topfivePanel.add(Box.createRigidArea(new Dimension(0,10)));
topfivePanel.add(oneLabel);
topfivePanel.add(one);
topfivePanel.add(twoLabel);
topfivePanel.add(two);
topfivePanel.add(threeLabel);
topfivePanel.add(three);
topfivePanel.add(fourLabel);
topfivePanel.add(four);
topfivePanel.add(fiveLabel);
topfivePanel.add(five);
topfivePanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
    createLineBorder(Color.BLUE), topfivePanel.getBorder()));

topthreePanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
    createLineBorder(Color.BLUE), toppthreePanel.getBorder()));
topfivePanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
    createLineBorder(Color.BLUE), topfivePanel.getBorder()));

// JPanel som inneholder top3 og top5
JPanel topPanel = new JPanel();
topPanel.setLayout(new BoxLayout(topPanel,BoxLayout.LINE_AXIS));
topPanel.setBackground(col);
topPanel.add(topthreePanel);
topPanel.add(Box.createHorizontalGlue());
topPanel.add(Box.createRigidArea(new Dimension(10,0)));
topPanel.add(topfivePanel);

topthreePanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
    createLineBorder(Color.BLUE), toppthreePanel.getBorder()));
topfivePanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
    createLineBorder(Color.BLUE), topfivePanel.getBorder()));

// JPanel med JButton for å få oversikt over alle proteintreff
JPanel bottomPanel = new JPanel();
bottomPanel.setLayout(new BoxLayout(bottomPanel,BoxLayout.PAGE_AXIS));
bottomPanel.setBackground(col);

JLabel geneLabel = new JLabel("Possible genes found:");
geneArea = new JTextArea();
geneArea.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
    createLineBorder(Color.cyan), geneArea.getBorder()));
geneArea.setEditable(false);
geneArea.setMaximumSize(new Dimension(80,20));
JPanel genePanel = new JPanel();
genePanel.setLayout(new BoxLayout(genePanel, BoxLayout.LINE_AXIS));
genePanel.setBackground(col);
genePanel.setBorder(BorderFactory.createEmptyBorder(2,2,2,2));
genePanel.add(geneLabel);
genePanel.add(Box.createRigidArea(new Dimension(10,0)));
genePanel.add(geneArea);

JLabel proteinLabel = new JLabel("Possible proteins found:");
proteinArea = new JTextArea();
proteinArea.setEditable(false);
proteinArea.setMaximumSize(new Dimension(80,20));
JPanel proteinPanel = new JPanel();
proteinPanel.setLayout(new BoxLayout(proteinPanel, BoxLayout.LINE_AXIS));
proteinPanel.setBackground(col);
proteinPanel.setBorder(BorderFactory.createEmptyBorder(2,2,2,2));
proteinPanel.add(proteinLabel);
proteinPanel.add(Box.createRigidArea(new Dimension(10,0)));
proteinPanel.add(proteinArea);

JButton bioTermsButton = new JButton("All bioterm");
bioTermsButton.addActionListener(this);
bottomPanel.add(genePanel);
bottomPanel.add(Box.createRigidArea(new Dimension(0,10)));
bottomPanel.add(proteinPanel);
bottomPanel.add(Box.createRigidArea(new Dimension(0,10)));
bottomPanel.add(bioTermsButton);
bottomPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
bottomPanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(Color.BLUE), bottomPanel.getBorder()));
contentPane.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
contentPane.add(panelUpper);
contentPane.add(Box.createRigidArea(new Dimension(0,5)));
contentPane.add(radioButtonPanelContainer);
contentPane.add(Box.createRigidArea(new Dimension(0,5)));
contentPane.add(topPanel);
contentPane.add(Box.createRigidArea(new Dimension(0,5)));
contentPane.add(bottomPanel);

try {
    UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    //UIManager.setLookAndFeel("com.sun.java.swing.plaf.mac");
} catch (Exception e) {
}
frame.setVisible(true);
System.out.println(bioTermsButton.size());
System.out.println(bioTermsButton.getLocation());
System.out.println(bottomPanel.size());
}
/*
 *Denne metoden setter inn de fem mest
 *relevante biotermene i GUIen etter indeks i
 */
public void insertTopFiveRelevantBioTerms(String term, int i) {
    if (i==0) {
        one.insert(term, 0);
    } else {
        two.insert(term, 0);
    }
    if (i==2) {
        three.insert(term, 0);
    } else {
        four.insert(term, 0);
    }
    if (i==4) {
        five.insert(term, 0);
    }
}

/*
* Denne metoden setter inn de tre mest relevante generelle termene.
*/
public void insertTopThreeRelevantTerms(String term, int i) {
    if (i==0) {
        first.insert(term, 0);
    } else {
        second.insert(term, 0);
    }
    if (i==2) {
        third.insert(term, 0);
    } else {
        //uinteressant i
    }
}

/*
* Denne metoden setter inn antallet identifiserte gen og protein i GUIen.
*/
public void insertHits(int countGeneHits, int countProteinHits) {
    Integer i = new Integer(countGeneHits);
    String geneHits = i.toString();
    geneArea.insert(geneHits, 0);
    Integer j = new Integer(countProteinHits);
    String proteinHits = j.toString();
    proteinArea.insert(proteinHits, 0);
589 }
package master;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import org.xml.sax.helpers.DefaultHandler;

/*
 * The Main Class. This class initiates the GUI and starts off
 * the whole chain of events.
 */

/**
 * @author skuland
 */

public class Main extends DefaultHandler{
    private FileOutputStream out; // declare a file output object
    private PrintStream p; // declare a print stream object
    private String pubmedid, a = "", b = "", term, retmode = "xml",
    pubmedIDString;
    private int reldate, retmax;
    private final int retstart = 0;
    public File abstracts, pubmedidtxt, eSearchResults, eSearchResultsCopy,
    eFetchResultsCopy, eFetchResults;
    private FileReader fr;
    private BufferedReader br;
    private URL url, urlFetch;
    private URLConnection urlConnection, urlConnectionFetch;
    private DataOutputStream printout;
    private InputStream inputStream, inputStreamFetch;
    private InputStreamReader inputStreamReader, inputStreamReaderFetch;
    private BufferedReader bufferedReader, bufferedReaderFetch;
    private FileWriter filewriter, filewriterFetch;
    private BufferedWriter bufferedwriter, bufferedwriterFetch;
    private PrintWriter printwriter, printwriterFetch;
    private ArrayList pubmedIDList, abstractTextList;
    private BufferedReader geneReader, geneReader2;
    private PrintWriter geneWriter;

public Main(String searchTerm, GUI g, boolean useStemming, boolean removeNumbers) 
{ 
    retmax(10); 
    reldate(100); 
    String x = searchTerm; 
    pubmedIDString = eSearch(x); 
    abstractTextList = eFetch(pubmedIDString); 
    // Dersom jeg ønsker å parse uniref100.xml 
    // Parse p = new Parse(); 
    // p.parseProteinNamesXMLFile(); 
    Tokenize tok = new Tokenize(pubmedIDString, abstractTextList, g, useStemming, removeNumbers); 
    // long now = System.currentTimeMillis(); 
    // ProduceHashTables pht = new ProduceHashTables(); 
    // long after = System.currentTimeMillis(); 
    // System.out.println((double)(after-now)/1000); 
} 


public ArrayList eFetch(String pubmedIDString) 
{ 
    try 
    
    urlFetch = new URL(eFetchURL); 
    urlConnectionFetch = urlFetch.openConnection(); 
    urlConnectionFetch.setDoInput(true); 
    urlConnectionFetch.setDoOutput(true); 
    urlConnectionFetch.setRequestProperty("Content-Type", "application/x-www-form-urlencoded"); 
    urlConnectionFetch.connect(); 
    inputStreamFetch = urlConnectionFetch.getInputStream(); 
    inputStreamReaderFetch = new InputStreamReader(inputStreamFetch); 
    BufferedReader bufferedReaderFetch = new BufferedReader(inputStreamReaderFetch); 
    Parse pa = new Parse(); 
    //abstractTextList = pa.returnAbstract(eFetchResults); 
    abstractTextList = pa.returnAbstract(inputStreamFetch); 
} 

catch(Exception e)
public String eSearch(String term) {
    this.term = term;

    try {
            esearch.fcgi?db=pubmed&term=" + term + "&retdate=" + reldate + "&retstart=" + retstart + "&retmax=" + retmax + "&retmode=" + retmode;
        System.out.println(eSearchURL);
        url = new URL(eSearchURL);
        urlConnection = url.openConnection();
        urlConnection.setDoInput(true);
        urlConnection.setDoOutput(true);
        urlConnection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
        urlConnection.connect();
        inputStream = urlConnection.getInputStream();
        inputStreamReader = new InputStreamReader(inputStream);
        bufferedReader = new BufferedReader(inputStreamReader);
        Parse ga = new Parse();
        pubmedIDList = ga.returnPubMedIDs(inputStream);
        for (int j = 0; j < pubmedIDList.size(); j++)
            if (j < pubmedIDList.size() - 1) {
                a = (String) pubmedIDList.get(j) + ",";
                b = b + a;
            }
            else 
                a = (String) pubmedIDList.get(j);
                b = b + a;
        pubmedIDString = b;
    }
    catch (Exception e) 
        e.printStackTrace();
    }

    return pubmedIDString;
}
{  return searchTerm;
}

public void retmax(int j)
{
  retmax = j;
}

public void reldate(int i)
{
  reldate = i;
}

public static void main(String args[])
{
  GUI gui = new GUI();
  //Main main = new Main();
}

private void geneDictionaryToLowerCase()
{
  try
  {
    geneReader = new BufferedReader(new FileReader(new File("geneSorted.txt")));
    geneReader2 = new BufferedReader(new FileReader(new File("geneSorted2.txt")));
  }catch(FileNotFoundException fe){fe.printStackTrace();}
  try{
    geneWriter = new PrintWriter(new BufferedWriter(new FileWriter(new File("geneDictionary.txt")), true));
  }catch(IOException io){io.printStackTrace();}
  String line = "";
  int countA = 0;
  int countB = 0;
  try{
    while( (line = geneReader.readLine()) != null )
    {
      countA++;
      if(line.length() > 0)
      {
        String newline = line.toLowerCase();
        geneWriter.println(newline);
        //geneWriter.println(countB);
        countB++;
      }
    }
  }
  catch(Exception e){}
  try{
    while( (line = geneReader2.readLine()) != null )
  {  
  
  }
  }
  catch(Exception e){}
countA++;

if(line.length() > 0)
{
    String newline = line.toLowerCase();
geneWriter.println(newline);
    //geneWriter.println(countB);
    countB++;
}

} catch(Exception e){
System.out.println("Verdi til countA: "+countA);
System.out.println("Verdi til countB: "+countB);
System.out.println("geneDictionary er ferdig:");
}
package master;

import java.io.BufferedWriter;
import java.io.CharArrayWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.sax.Attributes;
import javax.xml.sax.InputSource;
import javax.xml.sax.Locator;
import javax.xml.sax.SAXException;
import javax.xml.sax.SAXNotRecognizedException;
import javax.xml.sax.SAXNotSupportedException;
import javax.xml.sax.SAXParseException;
import javax.xml.sax.XMLReader;
import javax.xml.sax.helpers.DefaultHandler;

/*
* This class provides the parsing of all
* XML files used in the system.
*/

public class Parse extends DefaultHandler{

public String abstracttext;
private SAXParser parser2;
public File parsed_abstract;
public FileWriter fw;
public File uniref100;
public File proteinnames;
public BufferedWriter bw;
public PrintWriter pw;
private Locator locator;
public PrintWriter pw2;
public FileWriter fw2;
public BufferedWriter bw2;
private int counter = 0;
private StringBuffer sb;
private XMLReader parser;
public ArrayList pmidList, abstractTextList;
private CharArrayWriter charwriter;
boolean ampersand = false;
boolean b = false;

/*@author skuland*/
}
/*Denne skaper en SAXParser fra SAXParserFactory.*/

public Parse() {
    pmidList = new ArrayList();
    abstractTextList = new ArrayList();
    sb = new StringBuffer();

    try {
        uniref100 = new File("C:\uniref100.xml\uniref100xml");
    }
    catch (Exception e) {
        e.printStackTrace();
        System.out.println("feil");
    }

    //Denne metoden prøvde jeg først med SAXParser fra parserfactory, men det gav problemer med UTF8 malformed char
    try {
        SAXParserFactory parserFactory = SAXParserFactory.newInstance();
        parserFactory.setFeature("http://xml.org/sax/features/namespaces", true);
        parserFactory.setFeature("http://xml.org/sax/features/namespace-prefixes", false);
        parserFactory.setFeature("http://xml.org/sax/features/unicode-normalization-checking", true);
        parserFactory.setFeature("http://xml.org/sax/features/validation", true);
        parser2 = parserFactory.newSAXParser();
        parser2.getXMLReader().setContentHandler(this);
        parser2.getXMLReader().setDTDHandler(this);
        parser2.getXMLReader().setErrorHandler(this);
        parser2.getXMLReader().setEntityResolver(this);

        } catch (SAXNotRecognizedException e) {
        e.printStackTrace();
    } catch (SAXNotSupportedException e) {
        e.printStackTrace();
    } catch (ParserConfigurationException e) {
        e.printStackTrace();
    } catch (SAXException e) {
        e.printStackTrace();
    }
}

/*Metode som ble kalt for å parse den initiale +2.6GB XML protein fila*/

public void parseProteinNamesXMLFile() {
    try {
        parser2.parse(uniref100, this);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
/*
* Metoden som kalles fra Main som returnerer
* pubmedIDene som en ArrayList
*/
public ArrayList returnPubMedIDs(InputStream is) {
    try {
        parser2.parse(is, this);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return pmidList;
}

/*
* Metoden som kalles fra Main og som returnerer
* abstractene som en ArrayList
*/
public ArrayList returnAbstract(InputStream is) {
    try {
        parser2.parse(is, this);
    } catch (Exception e) {
        System.out.println("exception "+e.toString());
        e.printStackTrace();
    }
    return abstractTextList;
}

public void startDocument() {
    //System.out.println("Starten av dokumentet");
}

int number=0;

/*
* Standardmetode som kalles hver gang en XML
* startnode møtes
*/
public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
    try {
        //Id-elementet i eSearchResults.xml
        if(localName == "Id") {
            counter = 1;
            //System.out.println("pmid");
        }
    }
// name-elementet i uniref100.xml
  else if(localName == "name")
    {
      counter = 3;
    }

  // AbstractText-elementet i eFetchResults.xml
  else if(localName == "AbstractText")
  {
    // abstractFinished = false;
    number++;  
    // abstractTextElementFinished = true;
    counter = 2;
    // System.out.println("AbstractText begins: ");
  }
  else
  {
    counter=0;
  }
}

try{
  for(int i = start; i <= start+length-1; i++)
    {
      sb.append(ch[i]);
    }
  String pubmedid = sb.toString();
  pmidList.add(pubmedid);
  sb.delete(0,sb.length());
  counter = 0;
}

} catch(Exception e)
  {
    e.printStackTrace();
    counter = 0;
  }
}

public void characters(char[] ch, int start, int length) throws SAXException
{
  // ID-elementet i eSearchResults.xml
  if(counter == 1)
  {
    try{
      for(int i = start; i <= start+length-1; i++)
        {
          sb.append(ch[i]);
        }
      String pubmedid = sb.toString();
      pmidList.add(pubmedid);
      sb.delete(0,sb.length());
      counter = 0;
    }

  }

  // Forsøge å fange en Malformed UTF-8 char  SAXParseException
  catch(Exception e)
  {
    e.printStackTrace();
    counter = 0;
  }
}

else if(counter == 2)
  {
    String s = new String(ch,start,length);
    sb.append(s);
  }

else if(counter == 3)
  {
    counter = 0;
  }
public void endElement(String uri, String localName, String qName) throws SAXException
{
    if (localName == "AbstractText")
    {
        String t = sb.toString();
        abstractTextList.add(t);
        sb.delete(0, sb.length());
    }
}

public void setDocumentLocator(Locator locator)
{
    this.locator = locator;
}

public InputSource resolveEntity(String publicId, String systemId) throws SAXException
{
    return null;
}

public void error(SAXParseException e) throws SAXException
{
    System.out.println("Super1feil"+e.toString());
}

public void fatalError(SAXParseException e) throws SAXException
{
    System.out.println("Super2feil"+e.toString());
}

public void warning(SAXParseException e) throws SAXException
{
    System.out.println("Super3feil"+e.toString());
}

/* (non-Javadoc)
 * @see javax.xml.parsers.SAXParserFactory#isFeatureSupported(java.lang.String)
 */
public boolean isFeatureSupported(String arg0) throws ParserConfigurationException, SAXNotRecognizedException, SAXNotSupportedException
{
    return false;
}

/* (non-Javadoc)
 * @see javax.xml.parsers.SAXParserFactory#newSAXParser() 
 */
public SAXParser newSAXParser() throws ParserConfigurationException, SAXException
{
    return null;
}
/* (non-Javadoc)
 * @see javax.xml.parsers.SAXParserFactory#setFeature(java.lang.String, boolean)
 */

public void setFeature(String arg0, boolean arg1) throws 
    ParserConfigurationException, SAXNotRecognizedException,
    SAXNotSupportedException
{
}
package master;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.text.StringCharacterIterator;
import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Tokenize {

    private String pmid, a;
    private String[] tokens, tok;
    private ArrayList abstractList, tokenizedAbstractsList, alist, toStopWords;
    private ArrayList proteinCandidates, geneCandidates;
    private File testread, testwrite;
    private FileReader fr;
    private BufferedReader br;
    private FileWriter fw;
    private BufferedWriter bw;
    private PrintWriter pw, print;

    public Tokenize(String pmid, ArrayList abstractList, GUI g, boolean useStemming, boolean removeNumbers) {
        this.pmid = pmid;
        this.abstractList = abstractList;
        alist = new ArrayList();
        tokenizedAbstractsList = new ArrayList();
        System.out.println(pmid);
        tokenizedAbstractsList = tokenize(abstractList);
        PreProcessing pp = new PreProcessing(pmid, tokenizedAbstractsList, g, useStemming, removeNumbers);
    }

    public Tokenize(String pmid, ArrayList abstractList, GUI g, boolean useStemming) {
        this.pmid = pmid;
        this.abstractList = abstractList;
        alist = new ArrayList();
        tokenizedAbstractsList = new ArrayList();
        System.out.println(pmid);
        tokenizedAbstractsList = tokenize(abstractList);
        PreProcessing pp = new PreProcessing(pmid, tokenizedAbstractsList, g, useStemming, false);
    }

    /*
     * Konstruktoren.
     */

    public Tokenize(String pmid, ArrayList abstractList, GUI g) {
        this.pmid = pmid;
        this.abstractList = abstractList;
        alist = new ArrayList();
        tokenizedAbstractsList = new ArrayList();
        System.out.println(pmid);
        tokenizedAbstractsList = tokenize(abstractList);
        PreProcessing pp = new PreProcessing(pmid, tokenizedAbstractsList, g, false, false);
    }

    /*
     * Denne metoden får inn ArrayList abstractList med
     * n abstract-text-objekter og returnerer dem tokenized
     */
```java
private ArrayList tokenize(ArrayList abstractList)
{
    //System.out.println("ABSTRACTLIST er " + abstractList.size());
    for (int y = 0; y < abstractList.size(); y++)
    {
        //Abstract-teksten
        String s = (String)abstractList.get(y);
        tokens = s.split("\s");

        //lægger alle tokens fra arrayet til en arraylist alist
        for (int i = 0; i < tokens.length; i++)
        {
            alist.add(i, tokens[i]);
        }
        //leser gjennom alle tokene i arraylisten
        for (int x = 0; x < alist.size(); x++)
        {
            //ord etterfulgt av punktum, fjerne punktum
            Pattern p = Pattern.compile(".*\u002E");
            Matcher m = p.matcher(((String)alist.get(x)));
            //". " fjerne rene punktum
            Pattern punktum = Pattern.compile("\u002E");
            Matcher mpunktum = punktum.matcher(((String)alist.get(x)));

            //kolon
            Pattern kolon = Pattern.compile("\u003A");
            Matcher mkolon = kolon.matcher(((String)alist.get(x)));

            //semitolon
            Pattern semikolon = Pattern.compile("\u003B");
            Matcher msemikolon = semikolon.matcher(((String)alist.get(x)));

            //bindestrek
            Pattern phyphen = Pattern.compile("\u002D");
            Matcher mhyphen = phyphen.matcher(((String)alist.get(x)));

            //komma
            Pattern pcomma = Pattern.compile("\u002C");
            Matcher mcomma = pcomma.matcher(((String)alist.get(x)));

            //ord etterfulgt av komma, fjerne komma
            Pattern pa = Pattern.compile(".*\u002C");
            Matcher ma = pa.matcher(((String)alist.get(x)));

            //Venstre og høyre parentes fjernes
            // (HAT), (GRP-R , NMB-R)
            Pattern pat = Pattern.compile("\u0028.*\u0029");
            Pattern pat1 = Pattern.compile("\u0028.*\u0029");
            Pattern pat2 = Pattern.compile(".*\u0029");
            Matcher mat = pat.matcher(((String)alist.get(x)));
            Matcher mat1 = pat1.matcher(((String)alist.get(x)));
            Matcher mat2 = pat2.matcher(((String)alist.get(x)));

            //klamperantenes
            Pattern psquarebracket1 = Pattern.compile(".*\u005B.*\u005D");
            Pattern psquarebracket2 = Pattern.compile("\u005B.*\u005D");
            Pattern psquarebracket3 = Pattern.compile(".*\u005D");
            Matcher msquare1 = psquarebracket1.matcher(((String)alist.get(x)));
            Matcher msquare2 = psquarebracket2.matcher(((String)alist.get(x)));
        }
    }
}
```
Matcher msquare3 = psquarebracket3.matcher(((String)alist.get(x)));  

// For å få treff på GRP-R (GPRReceptor) fjernes =  
// dette gjelder generelt gen-navn som BRS-3, fjern =  
Pattern p13 = Pattern.compile("\p{Upper}{2,4}\u002D(\p{Upper}){1}");  
Matcher m13 = p13.matcher(((String)alist.get(x)));  

// uttrykk som (-1)), altså flere non-whitespace charact. før/etter et  
// parantesuttrykk  
// ((3a) , (3a))- , d*(3a)-d , 16).  
Pattern parantheses1 = Pattern.compile("(\S+\u0028.*\u0029)");  
Pattern parantheses2 = Pattern.compile("(\S+\u0028.*\u0029)\S+");  
Pattern parantheses3 = Pattern.compile("(\S+\u0028.*\u0029\S+)\S+");  
Pattern parantheses4 = Pattern.compile("*.\u0029\S+");  
Matcher mparantheses1 = parantheses1.matcher(((String)alist.get(x)));  
Matcher mparantheses2 = parantheses2.matcher(((String)alist.get(x)));  
Matcher mparantheses3 = parantheses3.matcher(((String)alist.get(x)));  
Matcher mparantheses4 = parantheses4.matcher(((String)alist.get(x)));  

// eks. angina: 120; kolon, semikolon etter ord  
Pattern pangina = Pattern.compile(".*[\u003A|\u003B]+";  
Matcher mangina = pangina.matcher(((String)alist.get(x)));  

boolean hasMatched = false;  

// Fjerner tomme elementer ogrene punktum, kolon, semikolon, bindestrek -  
// innslag  
if(((String)alist.get(x)).length()== 0) || mpunktum.matches() || mkolon  
.matches() || msemikolon.matches() || mhyphen.matches())  
{  
    alist.remove(x);  
x--;  
}  

else if(msquarel.matches())  
{  
    //System.out.println(((String)alist.get(x));  
    String newstring = ((String)alist.get(x)).substring(1, ((String)alist.  
    .get(x)).length() -1);  
    alist.remove(x);  
    alist.add(x, newstring);  
x--;  
}  

else if(msquare2.matches())  
{  
    //System.out.println("jlooo"+(String)alist.get(x));  
    String newstring = ((String)alist.get(x)).substring(1, ((String)alist.  
    .get(x)).length());  
    alist.remove(x);  
    alist.add(x, newstring);  
x--;  
}  

else if(msquare3.matches())  
{  
    //System.out.println("jlooo"+(String)alist.get(x));  
    String newstring = ((String)alist.get(x)).substring(0, ((String)alist.  
    .get(x)).length()-1);  
    alist.remove(x);  
    alist.add(x, newstring);  
}
x--; } 

else if (m13.matches()) 
{
  //System.out.println("GRS-3 treff her");
  //System.out.println((String)alist.get(x));
  String newstring = ((String)alist.get(x)).replaceAll("\u002D", "");
  alist.remove(x);
  alist.add(x, newstring);
  x--; }

else if (mparatheses1.matches() || mparatheses2.matches() || mparatheses3.matches()) 
{
  //løser stringen og finner start og slutindeks, skriver substringen til ny string
  String par = (String)alist.get(x);
  StringCharacterIterator sci = new StringCharacterIterator(par, 0, par.length(), 0);
  int beginIndex=0;
  int endIndex=0;

  while(sci.getIndex() < sci.getEndIndex()){
    char c = sci.current();
    char n = sci.next();
    if(c=='(' && n != '(' )
    {
      beginIndex = sci.getIndex();
    }
    else if(c != ')' && n == ')
    {
      endIndex = sci.getIndex();
    }
  }
  String paranthesesGone = par.substring(beginIndex, endIndex);
  alist.remove(x);
  alist.add(x, paranthesesGone);
  x--; 
}

else if (mparatheses4.matches())
{
  String par = (String)alist.get(x);
  StringCharacterIterator sci = new StringCharacterIterator(par, 0, par.length(), 0);
  int beginIndex=0;
  int endIndex=0;
  while(sci.getIndex() < sci.getEndIndex())
  {
    char c = sci.current();
    char n = sci.next();
    if(c != ']' && n == ')
    {
      endIndex = sci.getIndex();
    }
  }
String str = par.substring(beginIndex, endIndex);
alist.remove(x);
x--;
}
else if(mat.matches())
{
  hasMatched = true;
  String oldstring = ((String)alist.get(x));
  String newstring = oldstring.substring(1, oldstring.length()-1);
  alist.remove(x);
  alist.add(x, newstring);
  x--;
}
else if(mat1.matches() && !hasMatched)
{
  hasMatched = true;
  String oldstring = ((String)alist.get(x));
  String newstring = oldstring.substring(1, oldstring.length());
  alist.remove(x);
  alist.add(x, newstring);
  x--;
}
else if(mat2.matches() && !hasMatched)
{
  hasMatched = true;
  String oldstring = ((String)alist.get(x));
  String newstring = oldstring.substring(0, oldstring.length());
  alist.remove(x);
  alist.add(x, newstring);
  x--;
}
else if(m.matches())
{
  String oldstring = ((String)alist.get(x));
  String newstring = oldstring.substring(0, oldstring.length()-1);
  alist.remove(x);
  alist.add(x, newstring);
  x--;
}
else if(ma.matches())
{
  String oldstring = ((String)alist.get(x));
  String newstring = oldstring.substring(0, oldstring.length()-1);
  alist.remove(x);
  alist.add(x, newstring);
  x--;
}
else if(mangina.matches())
{
  String oldstring = ((String)alist.get(x));
  String newstring = oldstring.substring(0, oldstring.length());
  alist.remove(x);
  alist.add(x, newstring);
  x--;
}
//alist skal her inneholde et abstract som er blitt tokenized
//blir cloned til blist som legges til tokenizedAbstractsList
ArrayList blist = (ArrayList)alist.clone();
tokenizedAbstractsList.add(blist);
alist.clear();
private void searchForPatterns(ArrayList a)
{
    String nextSeq = "NEXTSEQ:";
    proteinCandidates.add(nextSeq);
    for(int x = 0; x < a.size(); x++)
    {
        //FACTOR, RECEPTOR, ENZYME, PROTEIN
        //treff her gir sterk indikasjon på protein-navn
        Pattern p1 = Pattern.compile(".*\{factor\}\.*");
        Pattern p2 = Pattern.compile(".*\{receptor\}\.*");
        Pattern p3 = Pattern.compile(".*\{enzyme\}\.*");
        Pattern p4 = Pattern.compile(".*\{protein\}\.*");
        Matcher m1 = p1.matcher(((String)a.get(x)));
        Matcher m2 = p2.matcher(((String)a.get(x)));
        Matcher m3 = p3.matcher(((String)a.get(x)));
        Matcher m4 = p4.matcher(((String)a.get(x)));
        if (m1.matches() || m2.matches() || m3.matches() || m4.matches())
        {
            String this_token = ((String)a.get(x));
            String previous_token = "*";
            if(x > 0)
            {
                previous_token = ((String)a.get(x-1));
                proteinCandidates.add(previous_token);
            }
            else
            {
                previous_token = "*";
            }
            proteinCandidates.add(this_token);
            if(x < a.size()-1)
            {
                next_token = ((String)a.get(x+1));
                proteinCandidates.add(next_token);
            }
            else
            {
                next_token = "*";
            }
            proteinCandidates.add(nextSeq);
        }
    }
    //A FAMILY OF...
    //er interessert i ordet etter denne sekvensen
    Pattern p5 = Pattern.compile(".*\{a\}\.*");
    Pattern p6 = Pattern.compile(".*\{family\}\.*");
    Pattern p7 = Pattern.compile(".*\{of\}\.*");
// må teste for nullpointer, dvs. at indeks er size-3
Matcher m5 = p5.matcher(""svs.");
Matcher m6 = p6.matcher(""svs.");
Matcher m7 = p7.matcher(""svs.");
if(x<a.size()-3)
{
    m5 = p5.matcher(((String)a.get(x)));
    m6 = p6.matcher(((String)a.get(x+1)));
    m7 = p7.matcher(((String)a.get(x+2)));
}
else
{
    m5 = p5.matcher(""svs.");
    m6 = p6.matcher(""svs.");
    m7 = p7.matcher(""svs.");
}
if(m5.matches() && m6.matches() && m7.matches())
{
    String potential = (String)a.get(x+3);
    proteinCandidates.add(potential);
    //a.remove(x+3);
    a.remove(x+2);
    a.remove(x+1);
    a.remove(x);
    proteinCandidates.add(nextSeq);
}

//.....LIKE
Pattern p8 = Pattern.compile(".*\u002D(like)"svs.");
Matcher m8 = p8.matcher(""svs.");
if(x<a.size()-1)
{
    m8 = p8.matcher(((String)a.get(x)));
}
else
{
    m8 = p8.matcher(""svs.");
}
if(m8.matches())
{
    String this_token = ((String)a.get(x));
    String next_token = ((String)a.get(x+1));
    proteinCandidates.add(this_token);
    proteinCandidates.add(next_token);
    proteinCandidates.add(nextSeq);
}

//.....RELEASING
Pattern p9 = Pattern.compile(".*\u002D(releasing)"svs.");
Matcher m9 = p9.matcher(""svs.");
if(x<a.size()-1)
{
    m9 = p9.matcher(((String)a.get(x)));
}
else
{
    m9 = p9.matcher(""svs.");
}
if(m9.matches())
{
String this_token = ((String)a.get(x));
String next_token = ((String)a.get(x+1));
proteinCandidates.add(this_token);
proteinCandidates.add(next_token);
proteinCandidates.add(nextSeq);
}

// Er interessert i dette ordet (protein)
Pattern p10 = Pattern.compile(".*ase\).*\.(in)\);\nPattern p11 = Pattern.compile(".*\.(in)\);\nMatcher m10 = p10.matcher("\(\w\[\p{Lower}\]\){2,8}\));\nMatcher m11 = p11.matcher("\(\w\[\p{Lower}\]\){2,8}\));\nif (m10.matches() | | m11.matches())\n{\n  String this_token = ((String)a.get(x));\n  proteinCandidates.add(this_token);\n  proteinCandidates.add(nextSeq);\n}

// Hormone
Pattern p13 = Pattern.compile("(inhibit)\).*\(over\));\nMatcher m13 = p13.matcher("((String)a.get(x))-1));\nif (m13.matches())\n{\n  String this_token = ((String)a.get(x));\n  String next_token = ((String)a.get(x+1));\n  proteinCandidates.add(next_token);\n  proteinCandidates.add(nextSeq);\n}
Pattern p15 = Pattern.compile("(of|by)");
Pattern p16 = Pattern.compile("(mutation)" hạ);  
Pattern p17 = Pattern.compile("(of)");  
Pattern p18 = Pattern.compile("(over)?(expression)");  
Pattern p19 = Pattern.compile("(of)");  
Pattern p20 = Pattern.compile("(regulator)");  
Pattern p21 = Pattern.compile("(of)");  
Pattern p22 = Pattern.compile("(modification)");  
Pattern p23 = Pattern.compile("(of)");  
Pattern p24 = Pattern.compile("(control)");  
Pattern p25 = Pattern.compile("(of)");  
Pattern p26 = Pattern.compile("(production)");  
Pattern p27 = Pattern.compile("(of)");  
Pattern p28 = Pattern.compile("(enhancement)");  
Pattern p29 = Pattern.compile("(of)");  

Matcher m14 = p14.matcher("" hạ);  
Matcher m15 = p15.matcher("" hạ);  
Matcher m16 = p16.matcher("" hạ);  
Matcher m17 = p17.matcher("" hạ);  
Matcher m18 = p18.matcher("" hạ);  
Matcher m19 = p19.matcher("" hạ);  
Matcher m20 = p20.matcher("" hạ);  
Matcher m21 = p21.matcher("" hạ);  
Matcher m22 = p22.matcher("" hạ);  
Matcher m23 = p23.matcher("" hạ);  
Matcher m24 = p24.matcher("" hạ);  
Matcher m25 = p25.matcher("" hạ);  
Matcher m26 = p26.matcher("" hạ);  
Matcher m27 = p27.matcher("" hạ);  
Matcher m28 = p28.matcher("" hạ);  
Matcher m29 = p29.matcher("" hạ);  

if (x < a.size() - 2)
{
    m14 = p14.matcher((String)a.get(x));  
    m15 = p15.matcher((String)a.get(x+1));  
    m16 = p16.matcher((String)a.get(x));  
    m17 = p17.matcher((String)a.get(x+1));  
    m18 = p18.matcher((String)a.get(x));  
    m19 = p19.matcher((String)a.get(x+1));  
    m20 = p20.matcher((String)a.get(x));  
    m21 = p21.matcher((String)a.get(x+1));  
    m22 = p22.matcher((String)a.get(x));  
    m23 = p23.matcher((String)a.get(x+1));  
    m24 = p24.matcher((String)a.get(x));  
    m25 = p25.matcher((String)a.get(x+1));  
    m26 = p26.matcher((String)a.get(x));  
    m27 = p27.matcher((String)a.get(x+1));  
    m28 = p28.matcher((String)a.get(x));  
    m29 = p29.matcher((String)a.get(x+1));  
}

if (m14.matches() && m15.matches() || m16.matches() && m17.matches() || m18.matches() && m19.matches() || m20.matches() && m21.matches() || m22.matches() && m23.matches() || m24.matches() && m25.matches() || m26.matches() && m27.matches() || m28.matches() && m29.matches())
} || (m26.matches() && m27.matches()) || (m28.matches() && m29.matches()))
541 {
542     String proteincandidate = ((String)a.get(x+2));
543     proteinCandidates.add(proteincandidate);
544     proteinCandidates.add(nextSeq);
545 }
package master;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;
import java.util.regex.PatternSyntaxException;

/*
* Denne klassen skal preprosessere abstractene før de blir sendt for lookup. De har da allerede blitt tokenized, og skal gjennom fjerning av stoppord og eventuelt stemmes. De preprosesserte ferdige abstractene skal sendes til LookUp.
*/

/*
* @author skuland
*/

public class PreProcessing {

private String pmid;
private ArrayList tokenizedAbstractsList, removedStopwords, stemmedList, finalList;
private File frequency;
private HashMap tokensCounted;
private ArrayList sortedKeys, sortedFrequencies, tokFreq;
private PrintWriter frequencyWriter, print, print2, print3;
private FileWriter filewriter;
private BufferedWriter bufferedwriter;

/*
* Konstruktoren. Den får inn tokenized abstracts.
* Starter opp LookUp-klassen.
*/

public PreProcessing(String pmid, ArrayList tokenizedAbstractsList, GUI gui, boolean useStemming, boolean removeNumbers) {
    this.pmid = pmid;
    this.tokenizedAbstractsList = tokenizedAbstractsList;
    tokensCounted = new HashMap();
    sortedKeys = new ArrayList();
    sortedFrequencies = new ArrayList();
    tokFreq = new ArrayList();
    removedStopwords = removeStopwordsFromAbstracts(tokenizedAbstractsList);
    // Stemmer bare abstractene dersom brukeren ønsket dette i GUI
    if (useStemming) {
        stemmedList = stemAbstracts(removedStopwords);
        }

    else {

}
stemmedList = removedStopwords;

//finalList inneholder etter dette steget alle abstractene ferdig stemmet og stoppodfjernet
finalList = removeStopwordsFromAbstracts(stemmedList);
ArrayList z = finalList;
if(removeNumbers)
    { 
        finalList = removePurelyNumericalEntries(finalList);
    } else 
    { 
        finalList = z;
    }
ArrayList al = sortTokensByFrequency(finalList);
LookUp lu = new LookUp(pmid, finalList, gui, al, this, useStemming);

private ArrayList removeStopwordsFromAbstracts(ArrayList tokenizedAbstractsList) 
{
    double e = 0;
    double average = 0;
    Stopwords stop = new Stopwords();
    for(int k = 0; k<tokenizedAbstractsList.size();k++)
    
        ArrayList a = stop.toLowerCase((ArrayList)tokenizedAbstractsList.get(k));
        double d = stop.quote;
        e = e + d;
        ArrayList b = stop.removeStopwords(a);
        tokenizedAbstractsList.remove(k);
        tokenizedAbstractsList.add(k,b);
    }
    average = e / tokenizedAbstractsList.size();
    System.out.println("Gjennomsnittlig antall prosent fjernet fra abstractene: "+average);
    for(int z = 0; z<2;z++)
    { 
        ArrayList ar = (ArrayList)tokenizedAbstractsList.get(z);
    }
    return tokenizedAbstractsList;

private ArrayList stemAbstracts(ArrayList alist)
{
    Stemmer st = new Stemmer();
    //for alle abstract
for(int k = 0; k < alist.size(); k++){
  ArrayList xy = (ArrayList)alist.get(k);

  //for alle tokens i hvert abstract
  for(int i = 0; i < xy.size(); i++) {
    int b = ((String)xy.get(i)).length();
    for(int a = 0; a < b; a++) {
      st.add(((String)xy.get(i)).charAt(a));
    }
    st.stem();
    String stemmedToken = st.toString();
    if(!((String)xy.get(i)).equals(stemmedToken)) {
      xy.remove(i);
      xy.add(i, stemmedToken);
    }
  }
  alist.remove(k);
  alist.add(k, xy);
}
return alist;

private ArrayList removePurelyNumericalEntries(ArrayList finalList) {
  int county = 0;
  for(int z = 0; z < finalList.size(); z++) {
    ArrayList al = (ArrayList)finalList.get(z);
    try {
      for(int a = 0; a < al.size(); a++) {
        if (((String)al.get(a)).matches("[\d]+") && !((String)al.get(a)).matches("[\D]+") )
          county++;
        al.remove(a);
      }
    } catch (PatternSyntaxException e) {
      e.printStackTrace();
    }
    finalList.remove(z);
    finalList.add(z, al);
  }
  return finalList;
}

```java
public ArrayList sortTokensByFrequency(ArrayList finalList)
{
    for(int e = 0; e < finalList.size(); e++){
        ArrayList list = (ArrayList)finalList.get(e);
        Set s = tokensCounted.keySet();
        Iterator it = s.iterator();
        int n = 0;
        int[] frequency = new int[tokensCounted.size()];
        while(it.hasNext())
        {
            String m = ((String)it.next());
            Integer i = (Integer)tokensCounted.get(m);
            int a = i.intValue();
            frequency[n] = a;
            n++;
        }
        //Kaller quicksort på frequency-arrayet try{
            sort(frequency);
        } catch(Exception ex){ex.printStackTrace();}
        //må så iterere gjennom hele arrayet frequency og mappe frekvenser mot stringene
        Set newset = tokensCounted.keySet();
        Iterator newit = newset.iterator();
        //Fyller opp ArrayListene med tomme objekter
```
```java
Integer io = new Integer(0);
String xyz = "";
for(int xy = 0; xy < newset.size(); xy++) {
    sortedKeys.add(xyz);
    sortedFrequencies.add(io);
}

int test = 0;
boolean found = false;

for (Map.Entry<String, Integer> entry : tokensCounted.entrySet()) {
    test++;
    found = false;
    String m = entry.getKey();
    Integer i = entry.getValue();
    if (!found) {
        int a = i.intValue();
        if (!found) {
            // System.out.println("k er: "+k);
            // Hvis verdi a fra tokensCounted er lik verdi i frequency-array
            if (a == frequency[k]) {
                sortedKeys.set(k, m);
                sortedFrequencies.set(k, i);
                frequency[k] = 0;
                found = true;
            }
        }
    }
}

System.out.println("Tokfreq size: "+tokFreq.size());
ArrayList tokFreqClone = (ArrayList)tokFreq.clone();
tokFreq.clear();
```
return tokFreqClone;
}

private void testWrite(String s)
{
  try{
  }
  catch(Exception e){
    e.printStackTrace();
  }
}

private void createFile()
{
  try{
    frequencyPrintWriter = new PrintWriter(new BufferedWriter(new FileWriter(new File("frequency.txt"), false)));
  }
  catch(Exception e){
    e.printStackTrace();
  }
}

private void quickSort(int a[], int l, int r) throws Exception
{
  int M = 4;
  int i;
  int j;
  int v;
  if ((r-l)>M)
  {
    i = (r+1)/2;
    if (a[i]>a[l]) swap(a,l,i);    // Tri-Median Method!
    if (a[i]>a[r]) swap(a,l,r);
    if (a[i]>a[r]) swap(a,i,r);
    j = r-1;
    swap(a,i,j);
    i = 1;
    v = a[i];
    for(i;)
    {
      while(a[++i]<v);
      while(a[--j]>v);
    }
if (j<i) break;
    swap (a,i,j);

    } 
    swap(a,i,r-1);
    quickSort(a,l,j); 
    quickSort(a,i+1,r);
    }
    }
}
private void swap(int a[], int i, int j)
{
    int T;
    T = a[i];
    a[i] = a[j];
    a[j] = T;
}
private void insertionSort(int a[], int lo0, int hi0) throws Exception
{
    int i;
    int j;
    int v;
    for (i=lo0+1;i<hi0;i++)
    {
        v = a[i];
        j=i;
        while (((j>lo0) && (a[j-1]>v))
        {
            a[j] = a[j-1];
            j--; 
        }
        a[j] = v;
    }
}
public void sort(int a[]) throws Exception
{
    quickSort(a, 0, a.length - 1);
    insertionSort(a,0,a.length-1);
}
package master;
import java.util.ArrayList;

/*
 * Denne klassen inneholder logikk for å fjerne stopord fra abstractene. Stoppordlista er hentet fra Medline.
 */

/*@author skuland*/
public class Stopwords {
  private String pmid;
  private ArrayList tokens;
  private String[] stopwords = {"a", "about", "again", "all", "almost", "also", "although", "always", "among", "an", "and", "another", "any", "are", "as", "at", "be", "because", "been", "before", "being", "between", "both", "but", "by", "can", "could", "did", "do", "does", "done", "due", "during", "each", "either", "enough", "especially", "etc", "for", "found", "from", "further", "had", "has", "have", "having", "here", "how", "however", "i", "if", "im", "into", "is", "it", "its", "itself", "just", "kg", "km", "made", "mainly", "make", "may", "mg", "might", "ml", "mm", "most", "mostly", "must", "nearly", "neither", "no", "nor", "obtained", "of", "often", "on", "our", "overall", "perhaps", "quite", "rather", "really", "regarding", "seen", "seen", "several", "should", "show", "showed", "shown", "shows", "significantly", "since", "so", "some", "such", "than", "that", "the", "their", "they", "then", "there", "therefore", "these", "they", "this", "those", "through", "thus", "to", "upon", "use", "used", "using", "various", "very", "was", "we", "were", "what", "when", "which", "while", "with", "within", "without", "would", "abstract"};
  private ArrayList tokenList;
  private ArrayList finalTokenList;
  public double quote;

  /*Konstruktoren.
  */
  public Stopwords() {
  }

  /*Metoden som kalles fra andre klasser for å fjerne stopord.
  */
  public ArrayList removeStopwords(ArrayList tokenList) {
    int g = tokenList.size();
    int count = 0;
    boolean foundStopword = false;
    for (int i = 0; i < tokenList.size(); i++)
      if (foundStopword = false;
if(!foundStopword){
    for(int x = 0; x < stopwords.length; x++)
    {
        if(!foundStopword){
            // må passe på indeksen her, dersom den er null, kan den bli negativ
            if(((String)(tokenList.get(i))).equals(stopwords[x]))
            {
                count++;
                if(i > 0)
                {
                    tokenList.remove(i);
                    i--;
                    foundStopword = true;
                }
                else
                {
                    tokenList.remove(i);
                    i = -1;
                    foundStopword = true;
                }
            }
            }
        quote = (double)count/g;
    }
}
return tokenList;
}

/*
* Metode som gjør et string array til en ArrayList
*/
public ArrayList toArrayList(String[] tokens)
{
    for(int i = 0; i < tokens.length; i++)
    {
        tokenList.add(tokens[i]);
    }
    return tokenList;
}

/*
* Metode som setter en arraylist med abstracts til lowercase
*/
public ArrayList toLowerCase(ArrayList alist)
{
    for(int x = 0; x < alist.size(); x++)
    {
        String m = ((String)alist.get(x)).toLowerCase();
        alist.remove(x);
        alist.add(x,m);
    }
    return alist;
}
package master;

/*
 * Klasse som implementer Porter stemming algoritmen.
 */

/**
 * @author skuland
 */

/*
* Stemmer, implementing the Porter Stemming Algorithm
*
* The Stemmer class transforms a word into its root form. The input
* word can be provided a character at time (by calling add()), or at once
* by calling one of the various stem(something) methods.
*/

public class Stemmer{

    private char[] b;
    private int i, /* offset into b */
            i_end, /* offset to end of stemmed word */
        j, k;
    private static final int INC = 50;
    /* unit of size whereby b is increased */

    public Stemmer()
    {
        b = new char[INC];
        i = 0;
        i_end = 0;
    }

    /**
    * Add a character to the word being stemmed. When you are finished
    * adding characters, you can call stem(void) to stem the word.
    */
    public void add(char ch)
    {
        if (i == b.length)
        {
            char[] new_b = new char[i+INC];
            for (int c = 0; c < i; c++) new_b[c] = b[c];
            b = new_b;
        }
        b[i++] = ch;
    }

    /**
    * Adds wLen characters to the word being stemmed contained in a portion
    * of a char[] array. This is like repeated calls of add(char ch), but
    * faster.
    */
    public void add(char[] w, int wLen)
    {
        if (i+wLen >= b.length)
        {
            char[] new_b = new char[i+wLen+INC];
            for (int c = 0; c < i; c++) new_b[c] = b[c];
            b = new_b;
        }
        for (int c = 0; c < wLen; c++) b[i++] = w[c];
    }
for (int c = 0; c < wLen; c++) b[i++] = w[c];
}

/**
 * After a word has been stemmed, it can be retrieved by toString(),
 * or a reference to the internal buffer can be retrieved by getResultBuffer
 * and getResultLength (which is generally more efficient.)
 */
public String toString() { return new String(b,0,i_end); }

/**
 * Returns the length of the word resulting from the stemming process.
 */
public int getResultLength() { return i_end; }

/**
 * Returns a reference to a character buffer containing the results of
 * the stemming process. You also need to consult getResultLength()
 * to determine the length of the result.
 */
public char[] getResultBuffer() { return b; }

private final boolean cons(int i) {
    switch (b[i]) {
        case 'a': case 'e': case 'i': case 'o': case 'u': return false;
        case 'y': return (i==0) ? true : !cons(i-1);
        default: return true;
    }
}

private final int m() {
    int n = 0;
    int i = 0;
    while(true) {
        if (i > j) return n;
        if (! cons(i)) break; i++;
    }
    i++;
    while(true) {
        if (i > j) return n;
        if (cons(i)) break;
        i++;
    }
    i++;
    n++;
    while(true) {
        if (i > j) return n;
        if (! cons(i)) break;
    }
}
i++;  
}  
i++;  
}  

/* vowelinstem() is true <=> 0,...,j contains a vowel */  
private final boolean vowelinstem()  
{  
    int i;  
    for (i = 0; i <= j; i++) if (!cons(i)) return true;  
    return false;  
}  

/* doublec(j) is true <=> j,(j-1) contain a double consonant. */  
private final boolean doublec(int j)  
{  
    if (j < 1) return false;  
    if (b[j] != b[j-1]) return false;  
    return cons(j);  
}  

/* cvc(i) is true <=> i-2,i-1,i has the form consonant - vowel - consonant  
and also if the second c is not w,x or y. this is used when trying to  
restore an e at the end of a short word. e.g.  
cav(e), lov(e), hop(e), crim(e), but  
snow, box, tray. */  
private final boolean cvc(int i)  
{  
    if (i < 2 || !cons(i) || cons(i-1) || !cons(i-2)) return false;  
    {  
        int ch = b[i];  
        if (ch == 'w' || ch == 'x' || ch == 'y') return false;  
    }  
    return true;  
}  

private final boolean ends(String s)  
{  
    int l = s.length();  
    int o = j+1;  
    if (o < 0) return false;  
    for (int i = 0; i < l; i++) if (b[o+i] != s.charAt(i)) return false;  
    j = k-l;  
    return true;  
}  

/* setto(s) sets (j+1),...,k to the characters in the string s, readjusting k. */  
private final void setto(String s)  
{  
    int l = s.length();  
    int o = k-l+1;  
    for (int i = 0; i < l; i++) b[o+i] = s.charAt(i);  
    k = j+l;  
}  

/* r(s) is used further down. */  
private final void r(String s) { if (m() > 0) setto(s); }  

/* step1() gets rid of plurals and -ed or -ing. e.g.  
caresses -> caress  
ponies -> poni  
ties -> ti  
caress -> caress */
cats -> cat
feed -> feed
agreed -> agree
disabled -> disable
matting -> mat
mating -> mate
meeting -> meet
milling -> mill
messaging -> mess
meetings -> meet

*/
private final void step1()
{
if (b[k] == 's')
{
if (ends("sses")) k -= 2; else
if (ends("ies")) setto("i"); else
if (b[k-1] != 's') k--; }
if (ends("eed"))
{
if (m() > 0) k--; } else
if ((ends("ed") || ends("ing")) && vowelinstem())
{
k = j;
if (ends("at")) setto("ate"); else
if (ends("bl")) setto("ble"); else
if (ends("iz")) setto("ize"); else
if (doublec(k))
{
int ch = b[k];
if (ch == 'l' || ch == 's' || ch == 'z') k++; }
else if (m() == 1 && cvc(k)) setto("e");
}
} /* step2() turns terminal y to i when there is another vowel in the stem. */
private final void step2()
{
if (ends("y") && vowelinstem()) b[k] = 'i';
}
/* step3() maps double suffixes to single ones. so -ization (= -ize plus -ation) maps to -ize etc. note that the string before the suffix must give m() > 0. */
private final void step3()
{
if (k == 0) return; /* For Bug 1 */ switch (b[k-1])
{
case 'a': if (ends("ational")) r("ate"); break; } 
if (ends("tional")) r("tion"); break; 
break;
case 'c': if (ends("enci")) r("ence"); break; }
if (ends("anci")) r("ance"); break; }
break;
case 'e': if (ends("izer")) r("ize"); break; }
break;
case 'l': if (ends("bli")) r("ble"); break; }
if (ends("alli")) r("all"); break; }
if (ends("entli")) r("ent"); break; }
if (ends("eli")) r("e"); break; }
if (ends("ousli")) r("ous"); break; }
break;
case 'o': if (ends("ization")) r("ize"); break; }
}
private final void step4() { switch (b[k])
    {
    case 'e': if (ends("icate")) { r("ic"); break; }
    if (ends("ative")) { r("a"); break; }
    if (ends("alize")) { r("al"); break; }
    break;
    case 'i': if (ends("iciii") { r("ic"); break; }
    if (ends("iiii") { r("ive"); break; }
    break;
    case 'l': if (ends("icall") { r("ic"); break; }
    if (ends("full") { r("ful"); break; }
    break;
    case 's': if (ends("ness") { r("n"); break; }
    break;
    }
    }
    
    /* step5() deals with -ic-, -full, -ness etc. similar strategy to step3. */
    private final void step5() { switch (b[k])
        {
        case 'e': if (ends("icate") { r("ic"); break; }
        if (ends("ative") { r("a"); break; }
        if (ends("alize") { r("al"); break; }
        break;
        case 'i': if (ends("iciii") { r("ic"); break; }
        if (ends("iiii") { r("ive"); break; }
        break;
        case 'l': if (ends("icall") { r("ic"); break; }
        if (ends("full") { r("ful"); break; }
        break;
        case 's': if (ends("ness") { r("n"); break; }
        break;
        }
        }
        
    /* step5() takes off -ant, -ence etc., in context <c>v<vcv<vc>. */
    private final void step5() { if (k == 0) return; /* for Bug 1 */
        switch (b[k-1])
        {
        case 'a': if (ends("al") break; return;
        case 'c': if (ends("ance") break; return;
        if (ends("ence") break; return;
        case 'e': if (ends("er") break; return;
        case 'i': if (ends("ic") break; return;
        case 'l': if (ends("ible") break; return;
        if (ends("ible") break; return;
        case 'n': if (ends("ant") break;
        if (ends("ement") break;
        if (ends("ment") break;
        /* element etc. not stripped before the n */
        if (ends("ent") break; return;
        case 'o': if (ends("ion") && j >= 0 && (b[j] == 's' || b[j] == 't'))
            break;
            /* j >= 0 fixes Bug 2 */
    if (ends("ou") break; return;
    /* takes care of -ous */
    case 's': if (ends("ism") break; return;
    case 't': if (ends("ate") break;
    if (ends("iti") break; return;
    case 'u': if (ends("ous") break; return;
    case 'v': if (ends("ive") break; return;
    case 'z': if (ends("ize") break; return;
    default: return;
    }
    if (m() > 1) k = j;
private final void step6()
{
    j = k;
    if (b[k] == 'e')
    {
        int a = m();
        if (a > 1 || a == 1 && !cvc(k-1)) k--;
    }
    if (b[k] == 'l' && doublec(k) && m() > 1) k--;
}

/step6() removes a final -e if m() > 1. */

/\* Stem the word placed into the Stemmer buffer through calls to add(). *
/\* Returns true if the stemming process resulted in a word different from the input. You can retrieve the result with *
/\* getResultSet()/getResultBuffer() or toString(). */

/public void stem()
{
    k = i - 1;
    if (k > 1) { step1(); step2(); step3(); step4(); step5(); step6(); }
    i_end = k+1; i = 0;
}

```java
package master;

import java.util.ArrayList;

/*
 * Denne klassen inneholder resultat ArrayList med alle gene- og protein-treff sortert etter frekvens.
 * @author skuland
 */

public class Result {

    private ArrayList resultList;
    private GUI gui;

    /*
     * Konstruktoren Result tar inn resultat ArrayListen og gui-pekeren
     */
    public Result(ArrayList resultList, GUI gui) {
        System.out.println("ER I RESULTKLASSEN");
        this.resultList = resultList;
        this.gui = gui;
        displayAllBioTermsInGUIWindow(resultList);
    }

    /*
     * Metode som setter inn resultatene i GUIen
     */
    public void displayAllBioTermsInGUIWindow(ArrayList resultList) {
        gui.receiveResultList(resultList);
    }
}
```
package master;

import java.util.ArrayList;

/*
 * Denne klassen beregner normalisert og invers frekvens
 * for alle termer i alle abstracts. Deretter beregnes
 * vekter for alle termene. De tre termene med høyest
 * vekt sender tilbake til LookUp.
 * N er antall dokumenter (abstracts) i systemet.
 */

/*
 * @author skuland
 */

public class Weight {

private ArrayList tokFreq;
private ArrayList normalizedFrequencies;
private ArrayList inverseFrequencies;
private ArrayList weightList, superWeightList;
private double normalizedFrequency = 0.0;
private double inverseDocumentFrequency = 0.0;
private int N;

public Weight() {
    normalizedFrequencies = new ArrayList();
    inverseFrequencies = new ArrayList();
    weightList = new ArrayList();
    superWeightList = new ArrayList();
}

/*
 * Konstruktør Weight.
 */

public ArrayList calculateFiveMostRelevantBioTerms(ArrayList gene, ArrayList protein) {
    double largestWeight = 0;
    double secondLargestWeight = 0;
    double thirdLargestWeight = 0;
    double fourthLargestWeight = 0;
    double fifthLargestWeight = 0;
    double a = 0;
    double b = 0;
    double c = 0;
double da = 0;
int filpIndex = 0;
int filuIndex = 0;
int folpIndex = 0;
int foluIndex = 0;
int tlpIndex = 0;
int tluIndex = 0;
int slpIndex = 0;
int sluIndex = 0;
int lpIndex = 0;
int luIndex = 0;

for(int u = 0; u < protein.size(); u++)
{
    ArrayList abstractWeights = (ArrayList)protein.get(u);
    for(int p = 0; p < abstractWeights.size(); p++)
    {
        if( ((Double)abstractWeights.get(p)).doubleValue() > fifthLargestWeight)
        {
            c = fifthLargestWeight;
            fifthLargestWeight = ((Double)abstractWeights.get(p)).doubleValue();
            filpIndex = p;
            filuIndex = u;
        }
        if( ((Double)abstractWeights.get(p)).doubleValue() > fourthLargestWeight)
        {
            da = fourthLargestWeight;
            fourthLargestWeight = ((Double)abstractWeights.get(p)).doubleValue();
            folpIndex = p;
            foluIndex = u;
            fifthLargestWeight = c;
        }
        if( ((Double)abstractWeights.get(p)).doubleValue() > thirdLargestWeight)
        {
            a = thirdLargestWeight;
            thirdLargestWeight = ((Double)abstractWeights.get(p)).doubleValue();
            tlpIndex = p;
            tluIndex = u;
            fourthLargestWeight = da;
        }
        if( ((Double)abstractWeights.get(p)).doubleValue() > secondLargestWeight)
        {
            b = secondLargestWeight;
            secondLargestWeight = ((Double)abstractWeights.get(p)).
            doubleValue();
            slpIndex = p;
            sluIndex = u;
            thirdLargestWeight = a;
        }
        if( ((Double)abstractWeights.get(p)).doubleValue() > largestWeight)
        {
            largestWeight = ((Double)abstractWeights.get(p)).doubleValue();
            lpIndex = p;
            luIndex = u;
            secondLargestWeight = b;
String mostRelevantTerm = (String)((ArrayList)tokFreq.get(luIndex*2)).get(lpIndex);
String secondMostRelevantTerm = (String)((ArrayList)tokFreq.get(sluIndex*2)).get(slpIndex);
String thirdMostRelevantTerm = (String)((ArrayList)tokFreq.get(tluIndex*2)).get(tlpIndex);
String fourthMostRelevantTerm = (String)((ArrayList)tokFreq.get(foluIndex*2)).get(folpIndex);
String fifthMostRelevantTerm = (String)((ArrayList)tokFreq.get(filuIndex*2)).get(filpIndex);

for(int g = 0; g < gene.size(); g++)
{
    ArrayList geneAbstractWeights = (ArrayList)protein.get(g);
    for(int ba = 0; ba < geneAbstractWeights.size(); ba++)
    {
        if( ((Double)geneAbstractWeights.get(ba)).doubleValue() > largestWeight)
            mostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
        else if( ((Double)geneAbstractWeights.get(ba)).doubleValue() > secondLargestWeight)
            secondMostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
        else if( ((Double)geneAbstractWeights.get(ba)).doubleValue() > thirdLargestWeight)
            thirdMostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
        else if( ((Double)geneAbstractWeights.get(ba)).doubleValue() > fourthLargestWeight)
            fourthMostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
        else if( ((Double)geneAbstractWeights.get(ba)).doubleValue() > fifthLargestWeight)
            fifthMostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
    }
}
ArrayList relevantBioTermsList = new ArrayList();
relevantBioTermsList.add(mostRelevantTerm);
relevantBioTermsList.add(secondMostRelevantTerm);
relevantBioTermsList.add(thirdMostRelevantTerm);
relevantBioTermsList.add(fourthMostRelevantTerm);
relevantBioTermsList.add(fifthMostRelevantTerm);
public ArrayList calculateNormalizedFrequency(ArrayList sortedKeys, ArrayList sortedFrequencies) {
    int maxFrequency = ((Integer)(sortedFrequencies.get(sortedFrequencies.size() - 1))).intValue();
    for(int i = 0; i < sortedKeys.size(); i++) {
        normalizedFrequency = (((double)(((Integer)sortedFrequencies.get(i)).intValue())) / ((double)maxFrequency));
        Double d = new Double(normalizedFrequency);
        normalizedFrequencies.add(d);
    }
    ArrayList normalizedFrequenciesClone = (ArrayList)normalizedFrequencies.clone();
    normalizedFrequencies.clear();
    return normalizedFrequenciesClone;
}

public int returnNumberOfTermOccurences(String term) {
    boolean isFoundHere = false;
    int occurrences = 0;
    for(int w = 0; w < tokFreq.size(); w = w + 2) {
        ArrayList sortedKeys = (ArrayList)tokFreq.get(w);
        if(!isFoundHere) {
            for(int r = 0; r < sortedKeys.size(); r++) {
                if(!isFoundHere) {
                    if(((String)sortedKeys.get(r)).equals(term)) {
                        occurrences++;
                        isFoundHere = true;
                    }
                }
            }
            isFoundHere = false;
        }
        return occurrences;
    }
}

/*
* Metode som beregner normalized frequency for generelle termer
* Dersom en term ikke opptrer i et dokument skal den normaliserte frekvensen være 0
*/

class Metode_1 {
    public ArrayList calculateNormalizedFrequency(ArrayList sortedKeys, ArrayList sortedFrequencies) {
        int maxFrequency = ((Integer)(sortedFrequencies.get(sortedFrequencies.size() - 1))).intValue();
        for(int i = 0; i < sortedKeys.size(); i++) {
            normalizedFrequency = ((double)(((Integer)sortedFrequencies.get(i)).intValue())) / ((double)maxFrequency);
            Double d = new Double(normalizedFrequency);
            normalizedFrequencies.add(d);
        }
        ArrayList normalizedFrequenciesClone = (ArrayList)normalizedFrequencies.clone();
        normalizedFrequencies.clear();
        return normalizedFrequenciesClone;
    }

    public int returnNumberOfTermOccurences(String term) {
        boolean isFoundHere = false;
        int occurrences = 0;
        for(int w = 0; w < tokFreq.size(); w = w + 2) {
            ArrayList sortedKeys = (ArrayList)tokFreq.get(w);
            if(!isFoundHere) {
                for(int r = 0; r < sortedKeys.size(); r++) {
                    if(!isFoundHere) {
                        if(((String)sortedKeys.get(r)).equals(term)) {
                            occurrences++;
                            isFoundHere = true;
                        }
                    }
                }
                isFoundHere = false;
            }
            return occurrences;
        }
    }

    /*
    * Metode som beregner inverse document frequency
    */
for generelle termer
*/

public ArrayList calculateInverseDocumentFrequency(ArrayList sortedKeys) {
    //idf_i = log (N/n_i)
    for (int i = 0; i < sortedKeys.size(); i++)
        int n_i = returnNumberOfTermOccurences((String)sortedKeys.get(i));
        double ratio = ((double)N) / ((double)n_i);
        inverseDocumentFrequency = Math.log(ratio);
        Double e = new Double(inverseDocumentFrequency);
        inverseFrequencies.add(e);
    ArrayList inverseFrequenciesClone = (ArrayList)inverseFrequencies.clone();
    inverseFrequencies.clear();
    return inverseFrequenciesClone;
}

/*Metode som returnerer superWeightList, som inneholder N *antall arraylister med vekter i stigende rekkefølge.
*/
public ArrayList calculateWeight(ArrayList tokFreq) {
    this.tokFreq = tokFreq;
    this.N = tokFreq.size()/2;
    //for alle abstracts, beregn frekvens og invers frekvens på alle termer
    //i arraylisten tokFreq er annenhvert objekt en sortedKeys og en sortedFrequencies-Arraylist
    for (int x = 0; x < tokFreq.size()−1; x=x+2)
        ArrayList sortedKeys = (ArrayList)tokFreq.get(x);
        ArrayList sortedFrequencies = (ArrayList)tokFreq.get(x+1);
        //dersom et abstract har fått minst ett protein- el gen-treff
        if(!(sortedFrequencies.size()==0)){
            ArrayList norm = calculateNormalizedFrequency(sortedKeys, sortedFrequencies);
            ArrayList inverse = calculateInverseDocumentFrequency(sortedKeys);
            for(int t = 0; t < norm.size(); t++)
                double weight = ((Double)norm.get(t)).doubleValue() * ((Double)inverse.get(t)).doubleValue();
                Double e = new Double(weight);
                //weightList får alle vektene til et abstract
                weightList.add(e);
            ArrayList weightListClone = (ArrayList)weightList.clone();
            superWeightList.add(weightListClone);
            weightList.clear();
        } //dersom abstractet ikke har fått treff( skjer på gen-treff i blant)
        else
            { double weight = 0;
                Double e = new Double(weight);
weightList.add(e);
ArrayList weightListClone = (ArrayList)weightList.clone();
superWeightList.add(weightListClone);
weightList.clear();
}
}
}

//System.out.println("superWeightList-size:"+
superWeightList.size());
ArrayList superWeightListClone = (ArrayList)superWeightList.clone();
superWeightList.clear();
return superWeightListClone;
}
*/
public ArrayList extractTopThreeMostRelevant(ArrayList superWeightList)
{
    double largestWeight = 0;
    double secondLargestWeight = 0;
    double thirdLargestWeight = 0;
    double b = 0;
    double a = 0;
    int tlpIndex = 0;
    int tluIndex = 0;
    int slpIndex = 0;
    int sluIndex = 0;
    int lpIndex = 0;
    int luIndex = 0;

    for(int u = 0; u < superWeightList.size(); u++)
    {
        ArrayList abstractWeights = (ArrayList)superWeightList.get(u);

        for(int p = 0; p < abstractWeights.size(); p++)
        {

            if( ((Double)abstractWeights.get(p)).doubleValue() > thirdLargestWeight)
            {
                a = thirdLargestWeight;
                thirdLargestWeight = ((Double)abstractWeights.get(p)).doubleValue();
                tlpIndex = p;
                tluIndex = u;
            }

            if( ((Double)abstractWeights.get(p)).doubleValue() > secondLargestWeight)
            {
                b = secondLargestWeight;
                secondLargestWeight = ((Double)abstractWeights.get(p)).doubleValue();
                slpIndex = p;
                sluIndex = u;
            }

            if( ((Double)abstractWeights.get(p)).doubleValue() > largestWeight)
            {
                largestWeight = ((Double)abstractWeights.get(p)).doubleValue();
                lpIndex = p;
                luIndex = u;
            }
        }
    }
}
String mostRelevantTerm = (String)((ArrayList)tokFreq.get(luIndex*2)).get(lpIndex);
String secondMostRelevantTerm = (String)((ArrayList)tokFreq.get(sluIndex*2)).get(slpIndex);
String thirdMostRelevantTerm = (String)((ArrayList)tokFreq.get(tluIndex*2)).get(tlpIndex);
ArrayList relevantList = new ArrayList();
relevantList.add(mostRelevantTerm);
relevantList.add(secondMostRelevantTerm);
relevantList.add(thirdMostRelevantTerm);
return relevantList;
package master;

import java.util.ArrayList;
import java.util.Hashtable;

/**
 * Denne klassen skal inneholde gen- og proteindictionaries.
 * Både stemmede og ustemmede versjoner skal kunne loades inn i minnet.
 */

public class Dictionary {
    public Hashtable geneDictionary;
    private ArrayList a;
    public Hashtable proteinDictionary_level_0, proteinDictionary_level_1,
                proteinDictionary_level_2, proteinDictionary_level_3,
                proteinDictionary_level_4, proteinDictionary_level_5,
                proteinDictionary_level_6, proteinDictionary_level_7,
                proteinDictionary_level_8, proteinDictionary_level_9;

    /*
     * Konstrukturen Dictionary som kaller ProduceHashTables-klassen
     * Gen og protein hash tables blir loaded inn i minnet.
     * useStemming bestemmer om stemmet eller ustemmet protein
     * hash table skal loades.
     */
    public Dictionary(boolean useStemming) {
        ProduceHashTables pht = new ProduceHashTables();
        this.geneDictionary = pht.createGeneHashTable();
        this.a = pht.createProteinHashtables(useStemming);
        this.proteinDictionary_level_0 = (Hashtable)a.get(0);
        this.proteinDictionary_level_1 = (Hashtable)a.get(1);
        this.proteinDictionary_level_2 = (Hashtable)a.get(2);
        this.proteinDictionary_level_3 = (Hashtable)a.get(3);
        this.proteinDictionary_level_4 = (Hashtable)a.get(4);
        this.proteinDictionary_level_5 = (Hashtable)a.get(5);
        this.proteinDictionary_level_6 = (Hashtable)a.get(6);
        this.proteinDictionary_level_7 = (Hashtable)a.get(7);
        this.proteinDictionary_level_8 = (Hashtable)a.get(8);
        this.proteinDictionary_level_9 = (Hashtable)a.get(9);
    }
}
package master;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;
import java.util.jar.JarFile;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/*
 * Klasse for å skape Hashtable-datastrukturene som skal benyttes
 * som proteindictionaries.
 *
 ***/
/*Author skuland
 * /
public class ProduceHashTables{
    private JarFile jf = null;
    private BufferedReader br, generader;
    private PrintWriter pw;
    private Hashtable nulltable;
    private Hashtable ht0, ht1, ht2, ht3, ht4, ht5, ht6, ht7, ht8, ht9;
    private Hashtable pointer0, pointer1, pointer2, pointer3, pointer4, pointer5,
        pointer6, pointer7, pointer8, pointer9;
    /*
 * Konstruktoren ProduceHashTables. Hash tables initialiseres.
 */
    public ProduceHashTables()
    {
        try{
            jf = new JarFile("prototype.jar");
        }catch(Exception e)
        {
        }
        ht0 = new Hashtable();
        ht1 = new Hashtable();
        ht2 = new Hashtable();
        ht3 = new Hashtable();
        ht4 = new Hashtable();
        ht5 = new Hashtable();
        ht6 = new Hashtable();
        ht7 = new Hashtable();
        ht8 = new Hashtable();
        ht9 = new Hashtable();
        pointer0 = new Hashtable();
    }
pointer1 = new Hashtable();
pointer2 = new Hashtable();
pointer3 = new Hashtable();
pointer4 = new Hashtable();
pointer5 = new Hashtable();
pointer6 = new Hashtable();
pointer7 = new Hashtable();
pointer8 = new Hashtable();
nulltable = new Hashtable();
}

public Hashtable createGeneHashTable()
{
    try
    {
// Get current classloader
ClassLoader cl = this.getClass().getClassLoader();
    URL geneURL = cl.getResource("final_dictionaries/geneDictionary.txt");
    URLConnection geneConnection = geneURL.openConnection();
    geneConnection.setDoInput(true);
    geneConnection.connect();
genereader = new BufferedReader(new InputStreamReader(geneConnection.
    getInputStream()));
    }
    catch(Exception fe){fe.printStackTrace();}
    Hashtable genetable = new Hashtable();
    Integer i = new Integer(0);
    String line = "";
    try
    {
        while( (line = genereader.readLine()) != null )
        {
            if(line.length() > 0)
            {
                genetable.put(line, i);
            }
        }
    }
    catch(IOException io){io.printStackTrace();}
    return genetable;
}

/*
*Denne metoden skaper gene hashtabellen. Returnerer hashtabellen.
*/
public ArrayList createProteinHashtables(boolean b)
{
    ClassLoader cl = this.getClass().getClassLoader();
    if(b){
        try
        {
            URL stProtURL = cl.getResource("final_dictionaries/
            stemmedProteinDictionary2.txt");
            URLConnection stProtConnection = stProtURL.openConnection();
            stProtConnection.setDoInput(true);
        }
        catch(Exception fe){fe.printStackTrace();}
    }
B Source code

```
118     stProtConnection.connect();
119     br = new BufferedReader(new InputStreamReader(stProtConnection.
120              getInputStream()));
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 int n = 11;
136 String line = "";
137 int length = 0;
138 int j = 0;
139 System.out.println(line.equals(null));
140 System.out.println(br.equals(null));
141 try{
142     //leser inn alle proteinnavnene, totalt 425X
143     while( (line = br.readLine()) != null )
144     {
145         String[] s = line.split("\s");
146         /*
147         * s inneholder nå alle tokens på denne linja i input-fila.
148         * Det er maksimalt 10 tokens på en linje som skal fordeles
149         * på HashTables
150         *
151         * Itererer over tokenene i et proteinnavn, fra 0 til max 9
152         * Alle de interne hashtabellene (pointer0-9) skal ha null
153         * på value, men det er ikke tilfelle for ht0 til ht9. Disse
154         * vil initielt ha null, men ved gjennomsleing av linje nr. 2
155         * må man ikke slette tidligere satt verdi (som kan være en
156         * hashtable med innslag på neste nivå).Fjerner paranterer,
157         * samt komma og punktum, kolon og semikolon
158         */
159         for(int k = 0; k < s.length; k++)
160         {
161             //0.nivå
162             if(k==0)
163             {
164                 //Dette ordet er allerede lagt inn i hashtabell på nivå 0
165                 if(ht0.containsKey(s[k]))
166                 {
167                     //Hashtable ha0 er den interne hashtabellen som er på value-plassen
168                     //til hashtable ht0. ha0 skal inneholde de nøkler som er nulige
169                     //kombinasjoner for ord nr. 2 i proteinnavnet
170                     Hashtable ha0 = (Hashtable)ht0.get(s[k]);
171                     ht0.put(s[k],ha0);
```
176    }  
177    // Nytt ord på nivå 0  
178    else  
179    {  
180      ht0.put(s[k], new Hashtable());  
181    }  
182  }  
183  // 1.nivå  
184  else if(k==1)  
185  {  
186    // Hvis dette ordet har blitt lagt til på 1.nivå før  
187    if(ht1.containsKey(s[k]))  
188    {  
189      Hashtable ha1 = (Hashtable)ht1.get(s[k]);  
190      ht1.put(s[k],ha1);  
191      Hashtable ha0 = (Hashtable)ht0.get(s[k-1]);  
192      ha0.put(s[k], nulltable);  
193      ht0.put(s[k-1],ha0);  
194    }  
195    // Dersom det er første gang dette ordet opptrer på 1.nivå  
196    else  
197    {  
198      // Legger til nøkkel på nivå 1 til hashtable på nivå 1  
199      ht1.put(s[k], new Hashtable());  
200      // Legger til nøkkel på 1.nivå til intern hashtable på nivå 0  
201      Hashtable ha0 = (Hashtable)ht0.get(s[k-1]);  
202      ha0.put(s[k], nulltable);  
203      ht0.put(s[k-1],ha0);  
204    }  
205  }  
206  // 2.nivå  
207  else if(k==2)  
208  {  
209    if(ht2.containsKey(s[k]))  
210    {  
211      Hashtable ha2 = (Hashtable)ht2.get(s[k]);  
212      ht2.put(s[k],ha2);  
213      Hashtable ha1 = (Hashtable)ht1.get(s[k-1]);  
214      ha1.put(s[k], nulltable);  
215      ht1.put(s[k-1],ha1);  
216    }  
217    else  
218    {  
219      ht2.put(s[k], new Hashtable());  
220      Hashtable ha1 = (Hashtable)ht1.get(s[k-1]);  
221      ha1.put(s[k], nulltable);  
222      ht1.put(s[k-1],ha1);  
223    }  
224  }  
225  // 3.nivå  
226  else if(k==3)  
227  {  
228    if(ht3.containsKey(s[k]))  
229    {  
230      Hashtable ha3 = (Hashtable)ht3.get(s[k]);  
231      ht3.put(s[k],ha3);  
232    }
Hashtable ha2 = (Hashtable)ht2.get(s[k-1]);
ha2.put(s[k], nulltable);
ht2.put(s[k-1],ha2);
}
else
{
 ht3.put(s[k], new Hashtable());
 Hashtable ha2 = (Hashtable)ht2.get(s[k-1]);
 ha2.put(s[k], nulltable);
 ht2.put(s[k-1],ha2);
}
//4. nívá
else if (k==4)
{
 if(ht4.containsKey(s[k]))
 {
  Hashtable ha4 = (Hashtable)ht4.get(s[k]);
  ht4.put(s[k],ha4);
  Hashtable ha3 = (Hashtable)ht3.get(s[k-1]);
  ha3.put(s[k], nulltable);
  ht3.put(s[k-1],ha3);
 }
 else
 {
  ht4.put(s[k], new Hashtable());
  Hashtable ha3 = (Hashtable)ht3.get(s[k-1]);
  ha3.put(s[k], nulltable);
  ht3.put(s[k-1],ha3);
 }
//5. nívá
else if (k==5)
{
 if(ht5.containsKey(s[k]))
 {
  Hashtable ha5 = (Hashtable)ht5.get(s[k]);
  ht5.put(s[k],ha5);
  Hashtable ha4 = (Hashtable)ht4.get(s[k-1]);
  ha4.put(s[k], nulltable);
  ht4.put(s[k-1],ha4);
 }
 else
 {
  ht5.put(s[k], new Hashtable());
  Hashtable ha4 = (Hashtable)ht4.get(s[k-1]);
  ha4.put(s[k], nulltable);
  ht4.put(s[k-1],ha4);
 }
//6. nívá
else if (k==6)
{
 if(ht6.containsKey(s[k]))
 {
  Hashtable ha6 = (Hashtable)ht6.get(s[k]);
  ht6.put(s[k],ha6);
 }
Hashtable ha5 = (Hashtable)ht5.get(s[k-1]);
ha5.put(s[k], nulltable);
ht5.put(s[k-1], ha5);
} else {
    ht6.put(s[k], new Hashtable());
    Hashtable ha5 = (Hashtable)ht5.get(s[k-1]);
    ha5.put(s[k], nulltable);
    ht5.put(s[k-1], ha5);
}
}
//7. nívá
else if(k==7) {
    if(ht7.containsKey(s[k])) {
        Hashtable ha7 = (Hashtable)ht7.get(s[k]);
        ht7.put(s[k], ha7);
        Hashtable ha6 = (Hashtable)ht6.get(s[k-1]);
        ha6.put(s[k], nulltable);
        ht6.put(s[k-1], ha6);
    } else {
        ht7.put(s[k], new Hashtable());
        Hashtable ha6 = (Hashtable)ht6.get(s[k-1]);
        ha6.put(s[k], nulltable);
        ht6.put(s[k-1], ha6);
    }
//8. nívá
else if(k==8) {
    if(ht8.containsKey(s[k])) {
        Hashtable ha8 = (Hashtable)ht8.get(s[k]);
        ht8.put(s[k], ha8);
        Hashtable ha7 = (Hashtable)ht7.get(s[k-1]);
        ha7.put(s[k], nulltable);
        ht7.put(s[k-1], ha7);
    } else {
        ht8.put(s[k], new Hashtable());
        Hashtable ha7 = (Hashtable)ht7.get(s[k-1]);
        ha7.put(s[k], nulltable);
        ht7.put(s[k-1], ha7);
    }
//9. nívá
else if(k==9) {
    if(ht9.containsKey(s[k])) {
        Hashtable ha9 = (Hashtable)ht9.get(s[k]);
    }
ht9.put(s[k], ha9);

Hashtable ha8 = (Hashtable) ht8.get(s[k - 1]);
ha8.put(s[k], nulltable);
ht8.put(s[k - 1], ha8);
}
else {
    ht9.put(s[k], new Hashtable());
    Hashtable ha8 = (Hashtable) ht8.get(s[k - 1]);
    ha8.put(s[k], nulltable);
    ht8.put(s[k - 1], ha8);
}
}
}
 System.out.println(ht0.size());
 System.out.println(ht1.size());
 System.out.println(ht2.size());
 System.out.println(ht3.size());
 System.out.println(ht4.size());
 System.out.println(ht5.size());
 System.out.println(ht6.size());
 System.out.println(ht7.size());
 System.out.println(ht8.size());
 System.out.println(ht9.size());
}
} catch (IOException ioe) { ioe.printStackTrace(); }
ArrayList listOfProteinHashTables = new ArrayList();
listOfProteinHashTables.add(ht0);
listOfProteinHashTables.add(ht1);
listOfProteinHashTables.add(ht2);
listOfProteinHashTables.add(ht3);
listOfProteinHashTables.add(ht4);
listOfProteinHashTables.add(ht5);
listOfProteinHashTables.add(ht6);
listOfProteinHashTables.add(ht7);
listOfProteinHashTables.add(ht8);
listOfProteinHashTables.add(ht9);
return listOfProteinHashTables;
}/*
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
while (line = br.readLine()) != null 
{
    if(line.length() > 0)
    {
        String[] s = line.split("\s");
        for(int i = 0; i < s.length;i++)
        {
            int b = s[i].length();
            for(int a = 0; a < b; a++){
                st.add((s[i]).charAt(a));
            }
            st.stem();
            String stemmedToken = st.toString();
            if(!(s[i]).equals(stemmedToken))
            {
                s[i] = stemmedToken;
            }
            pw.print(s[i] + " ");
        }
        pw.print("\r");
        pw.close();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
package master;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.PrintWriter;
import java.util.ArrayList;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXNotRecognizedException;
import org.xml.sax.SAXNotSupportedException;
import org.xml.sax.helpers.DefaultHandler;

/*
 * Deze klassen er laget for å forsøke å få testet systemet
 * mot ferdig taggede abstracts. Målet er å få tall på
 * precision og recall.
 */

/**
 * @author skuland
 *
 */

public class Test extends DefaultHandler{

private SAXParser parser2;
private int counter = 0;
private PrintWriter proteinWriter;
private PrintWriter pmidWriter;
private BufferedReader proteinReader;
private BufferedReader pmidReader;
private StringBuffer sb1, sb2;
private boolean insideAbstract = false;
private ArrayList pmidList, taggedProteinList;
private String a = "", b = "", pmidString = "";

/*
 * Konstruktoren skaper en SAXParser.
 */

public Test()
{
    pmidList = new ArrayList();
taggedProteinList = new ArrayList();
    try{
        SAXParserFactory parserFactory = SAXParserFactory.newInstance();
        parserFactory.setFeature("http://xml.org/sax/features/namespaces", true);
        parserFactory.setFeature("http://xml.org/sax/features/namespace-prefixes", false);
        parser2 = parserFactory.newSAXParser();
        parser2.getXMLReader().setContentHandler(this);
        parser2.getXMLReader().setDTDHandler(this);
    }
}
parser2.getXMLReader().setErrorHandler(this);
parser2.getXMLReader().setEntityResolver(this);
}

try {
e.printStackTrace();
} catch (SAXNotRecognizedException e) {
e.printStackTrace();
} catch (SAXNotSupportedException e) {
e.printStackTrace();
} catch (ParserConfigurationException e) {
e.printStackTrace();
} catch (SAXException e) {
e.printStackTrace();
}

//parseFile();
//readYapexTaggedProteins();
//readYapexPmids();

/*
 * Metode som leser inn alle taggede proteinavsn og puter de
 * i en arraylist taggedProteinList. Denne inneholder alle 1559
 * proteinene som er tagget i Yapex-treningssfila.
 */
public ArrayList readYapexTaggedProteins() {
    String line = "";
    try {
        proteinReader = new BufferedReader(new FileReader(new File("YAPEX_TAGGED_PROTEINS.txt")));
        proteinReader = new BufferedReader(new FileReader(new File("YAPEX_TAGGED_PROTEINS_TEST.txt")));
    } catch (Exception e) {}
    try {
        while ( (line = proteinReader.readLine()) != null )
        {
            if(line.length() > 0)
            {
                taggedProteinList.add(line);
            }
        }
    } catch (Exception e) {}
    System.out.println("taggedProteinList.size: "+taggedProteinList.size());
    return taggedProteinList;
}

/*
 * Metode som returnerer string med 99pmider som brukes til trening.
 */
public String readYapexPmids() {
    String line = "";
    try {
        pmidReader = new BufferedReader(new FileReader(new File("YAPEX_PMODS.txt")));
        pmidReader = new BufferedReader(new FileReader(new File("YAPEX_PMODS_TEST.txt")));
    } catch (Exception e) {}
    try {
        while ( (line = pmidReader.readLine()) != null )
        {
            if(line.length() > 0)
{ pmidList.add(line); }

catch(Exception e){
System.out.println("pmidlist: "+pmidList.size());
for(int j = 0; j<pmidList.size();j++)
{
if(j<pmidList.size()-1)
{ a = (String)pmidList.get(j)+ ",";
  b = b + a;
}
else
{ a = (String)pmidList.get(j);
  b = b + a;
}
}

//Her har jeg en pmidstring på form tall,tall,tall og størrelse 99
pmidString = b;
return pmidString;
}

/*Metode som initierer parsing av test XML fil*/
public void parseFile()
{
try{
  //File yapex = new File("master/yapex_ref_collection.txt");
  File yapex = new File("master/yapex_test_collection.txt");
  FileInputStream fis = new FileInputStream(yapex);
  //pmidWriter = new PrintWriter(new BufferedWriter(new FileWriter(new File("YAPEX_PMIDS_TEST.txt"),false)));
  //proteinWriter = new PrintWriter(new BufferedWriter(new FileWriter(new File("YAPEX_TAGGED_PROTEINS_TEST.txt"),false)));
  sb1 = new StringBuffer();
  sb2 = new StringBuffer();
  parser2.parse(fis, this);
  pmidWriter.close();
  proteinWriter.close();
  System.out.println("Antall pmider:*count");
  System.out.println("Antall protein i abstractene:"+proteinCount);
  System.out.println("Antall slutttagger PMID: "+ countPMIDEND);
  System.out.println("Antall slutttagger Protname: "+ countProtnameEnd);
  System.out.println("Counter == 1: "+county);
}
catch(Exception e)
{
public void startDocument()
{
    int count = 0;
    int proteinCount = 0;
    public void startElement(String uri, String localName, String qName, Attributes attributes)
        throws SAXException
    {
        try{
            if(localName == "PMID")
            {
                counter = 1;
                System.out.println("YAPEX PMID");
                count++;
            }
            else if(localName == "AbstractText")
            {
                insideAbstract = true;
            }
            else if(localName =="Protname" && insideAbstract)
            {
                proteinCount++;
                counter = 2;
            }
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }
    int county = 0;
    public void characters(char[] ch, int start, int length) throws SAXException
    {
        if(counter == 1)
        {
            county++;
            String s = new String(ch,start,length);
            sb1.append(s);
        }
        if(counter ==2)
        {
            String s = new String(ch,start,length);
            sb2.append(s);
        }
    }
    int countPMIDEND = 0;
    int countPronameEnd = 0;
    public void endElement(String uri, String localName, String qName) throws SAXException
    {
if(localName == "PMID")
{
    countPMIDEND++;
    counter = 0;
    String t = sb1.toString();
    pmidWriter.println(t);
    pmidWriter.println("\r\n");
    sb1.delete(0,sb1.length());
}
else if(localName =="Proname")
{
    countProtnameEnd++;
    counter = 0;
    String t = sb2.toString();
    proteinWriter.println(t);
    proteinWriter.println("\r");
    sb2.delete(0,sb2.length());
}
else if(localName == "AbstractText")
{
    insideAbstract = false;
}
}
package master;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collections;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;
import java.util.regex.Pattern;

/*
 * Class that has been used during construction
 * of the gene and protein dictionaries.
 * Contains mostly regular expression-logic.
 * */

public class ProcessDictionary {
  private File proteinnames, removedDuplicates, removedNumbers, testy, testfile;
  private File genenames, dictionary, genenames2, genenames3;
  private FileWriter fw, fw2, fw3, updatedFileWriter, dictionaryFileWriter;
  private BufferedWriter bw, bw2, bw3, updatedBufferedWriter, dictionaryBufferedWriter;
  private PrintWriter pw, pw2, pw3, printwriter, dictionaryPrintWriter;
  private FileReader fr, fr2, fr3, fr4, fr5, fr6;
  private BufferedReader br, br2, br3, br4, br5, br6;
  private Hashtable proteintable, proteintable2, genenamestable;
  private ArrayList list1, list2, sortProteinNamesList, sortGeneNamesList;
  private Calendar cal, cal2, cal3, cal4;

  /*
   * Konstruktoren brukes til å sette i gang
   * diverse processoringsmetoder.
   */
  public ProcessDictionary() {
    try {
      list1 = new ArrayList();
      list2 = new ArrayList();
      sortProteinNamesList = new ArrayList();
      sortGeneNamesList = new ArrayList();
      proteintable = new Hashtable();
      proteintable2 = new Hashtable();
genenamestable = new Hashtable();

try {
  e.printStackTrace();
}

cal3 = Calendar.getInstance();
long before2 = cal3.getTimeInMillis();

//removeEntries();
//testremoveEntries();
//processGeneNames();
//alphabeticallySortProteinNames();
//alphabeticallySortGeneNames();

magnus();
//removeDuplicateNames();
//removeEntries();
//alphabeticallySortProteinNames();
stemProteinNames();

cal4 = Calendar.getInstance();
long after2 = cal4.getTimeInMillis();
long time2 = after2 - before2;
System.out.println("Tidsforbruk: + (double)time2/1000 + " sekunder");
}

/*

*Denne metoden stemmer protein-navnene.
*/

private void stemProteinNames()
{
  BufferedReader br = null;
  PrintWriter pri = null;
  try{
    br = new BufferedReader(new FileReader(new File("proteinDictionary2.txt")));
    pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("stemmedProteinDictionary2.txt")),false));
  }
  catch(Exception e){}
  try{
    Stemmer st = new Stemmer();
    String line = "*";
    while( (line = br.readLine()) != null )
    {
      if(line.length() > 0)
      {
        String[] s = line.split("\\a");
        for(int i = 0; i < s.length;i++)
        {
          int b = s[i].length();
          for(int a = 0; a < b; a++)
          {
            st.add(s[i].charAt(a));
          }
          st.stem();
        }
        String stemmedToken = st.toString();
        if(!s[i].equals(stemmedToken))
      }
  }
{  
    s[i] = stemmedToken;
    }
    
    pri.print(s[i] + " ");
    }
    pri.print("\r");
    
    br.close();
    pri.close();
    }
    
    catch(Exception e)
    {
        e.printStackTrace();
    }
    
    /*
    *Regular-expression-metode.
    */
    public void magnus()
    {
        BufferedReader br = null;
        PrintWriter pri = null;
        try{
            br = new BufferedReader(new FileReader(new File("kournikova.txt")));
            pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("UPDATED - DICTIONARY.txt"),false)));
        }
        catch(Exception e){}
        //((String)al.get(a)).matches("[\d]*)

        int tell = 0;
        int county=0;
        StringBuffer sb = new StringBuffer();
        try{
            String line = "";
            String newline = "";
            while( (line = br.readLine()) != null )
            {
                if(line.length() > 0)
                {
                    county++;
                    //tagn).
                    if(line.matches(".*\\u0028\\u0029\"))
                    {
                        tell++;
                        line = line.substring(0, line.length()-2);
                    }
                    /*
                    *tagn )
                    if(line.matches(".*\\u0028\\u0029\")
                    {
                        tell++;
                        line = line.substring(0, line.length()-1);
                    }
pri.println(line);
/*
county++;
String[] s = line.split("\s");
if(s[0].equals("similarities") && s[1].equals("with"))
{
tell++;

for(int h = 2; h < s.length; h++)
{
    sb.append(s[h] + " ");
}
//sb.append(s[s.length-1]);
int lastindex = sb.length()-1;
System.out.println(lastindex);
if(lastindex > 0){
    sb.deleteCharAt(lastindex);
}
line = sb.toString();
System.out.println(line);
}
/*
*/
}
try{
    String line = ""
    while( (line=br6.readLine()) != null ){
        if(line.length() > 0)
        {
            sortGeneNamesList.add(line);
        }
    }
    br6.close();
    System.out.println("Size: "+sortGeneNamesList.size());
    Collections.sort(sortGeneNamesList);
    for(int i = 0; i< sortGeneNamesList.size(); i++)
```java
239     }
240     pw5.println((String)sortGeneNamesList.get(i));
241     pw5.close();
242
243     catch(Exception e)
244     {
245         e.printStackTrace();
246     }
247
248}
249
250
251                          /*
252                          * Metode som alfabetisk sorterer protein-navnene i tekstfila
253                          */
254  public void alphabeticallySortProteinNames()
255  {
256      BufferedReader br = null;
257      PrintWriter pri = null;
258      try{
259          br = new BufferedReader(new FileReader(new File("kournikova.txt")));
260          pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("UPDATED -
261                       DICTIONARY.txt"),false)));
262      }
263      catch(Exception e){}
264
265      try
266      {
267          String line = "";
268          while((line=br.readLine()) != null)
269          {
270              if(line.length() > 0)
271              {
272                  sortProteinNamesList.add(line);
273              }
274          }
275          br.close();
276          System.out.println("Size: "+sortProteinNamesList.size());
277          Collections.sort(sortProteinNamesList);
278          for(int i = 0; i< sortProteinNamesList.size(); i++)
279              {pri.println((String)sortProteinNamesList.get(i));
280          }
281          pri.close();
282      }
283      catch(Exception e)
284      {
285          e.printStackTrace();
286      }
287
288}
289
290                          /*
291                          * Metode for prosessering av gennavnene
292                          */
293  public void processGeneNames()
294  {
295      try{
296          Integer i = new Integer(0);
297          String line = "";
298          int size = 0;
299```
```java
int count = 0;
String withdrawn = ".*\p{Punct}(withdrawn).*";

while (line = br5.readLine()) != null {
    if (line.length() > 0)
        count++;
    boolean b = Pattern.matches(withdrawn, line);
    if (!b)
        genenamestable.put(line, i);
}

br5.close();
size = genenamestable.size();

// skrive genenamestable-objektet til fil
String genenameto_dictionary = genenamestable.toString();
dictionaryPrintWriter.println(genenameto_dictionary);
dictionaryPrintWriter.close();

Set keys = genenamestable.keySet();
Iterator it = keys.iterator();
while (it.hasNext())
    { String key = (String)it.next();
        pw3.println(key); } 
pw3.close();
System.out.println("Antall unike: " + size);
System.out.println("Fjernede elementer: " + (count - size));
System.out.println("Prosent av opprinnelig fil: " + (double)size/count);
}
catch (Exception e)
{
    e.printStackTrace();
}

/*
* Metode som fjerner duplikat-innslag
*/
public void removeDuplicateNames()
{
    BufferedReader br = null;
    PrintWriter pri = null;
    try{
        br = new BufferedReader(new FileReader(new File("UPDATED-DICTIONARY.txt")));
        pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("kournikova. txt")), false));
    }
catch (Exception e){}
    int size = 0;
    try{
```
```java
Integer i = new Integer(0);
String line;
int count = 0;
while (line = br.readLine()) != null {
    // den skriver ut annenhver linje pga. \r-tegnet, derfor må teste lengden større enn 0
    if (line.length() > 0) {
        count++;
        // System.out.println(line);
        proteintable.put(line, i);
    }
}
br.close();
size = proteintable.size();
Set keys = proteintable.keySet();
Iterator it = keys.iterator();
while (it.hasNext()) {
    String key = (String) it.next();
    pri.println(key);
}
pri.close();
System.out.println("Antall unike: "+ size);
System.out.println("Fjernede elementer: "+ (count - size));
System.out.println("Prosent av opprinnelig fil: "+ (double) size / count);
}
catch (Exception e) {
    e.printStackTrace();
}
*/
*/
 Regular expression-metode som fjerner diverse entries.
*/
private void removeEntries() {
    BufferedReader br = null;
    PrintWriter pri = null;
    try {
        br = new BufferedReader(new FileReader(new File("kournikova.txt")));
        pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("UPDATED-DICTIONARY.txt")), false));
    }
catch (Exception e) {
    }
    try {
        Integer i = new Integer(0);
        int countTotalLines = 0;
        int countLinesInUpdatedFile = 0;
        String line;
        while (line = br.readLine()) != null {
```
//den skriver ut annenhver linje tom pga. \r-tegnet, derfor må teste lengde større enn 0

if(line.length() > 0)
{
    countTotalLines++;

    String numerical = "[\\d]*";
    String numbers2 = "[\\D]*";
    String oneletter = "[a-zA-Z]{1}";
    String kda = "\d(1,4)\p{Punct}(0,1)\d(0,4)\s(kDa){1}\s(protein){1}";
    String predicted = "(PREDICTED:)\s(hypothetical)\.*";
    String uniref = ".\s(UniRef100)\s(entry)\.*";
    String hypothetical = "(H|h)(ypothetical)\.*";
    String predicted2 = "(PREDICTED:)\.*";
    String decimal = "\d*\p{Punct}d*";
    String similar = "(S|s)(imilar)\s(to)\.*";

    //hvis linja ikke er rene tall, skriv til fil

    if( !((line.matches(numbers1) && (! line.matches(numbers2)))) )
    {
        countLinesInUpdatedFile++;
        pri.println(line);
    }

    //hvis linja er en bokstav som "A"
    if(!line.matches(oneletter))
    {
        countLinesInUpdatedFile++;
        printwriter.println(line);
    }

    //Hvis linje er på form "23 kDa protein" el. "23.3 kDa protein"
    if(!line.matches(kda))
    {
        countLinesInUpdatedFile++;
        printwriter.println(line);
    }

    //Hvis linje på form "PREDICTED: hypothetical.."
    if(!line.matches(predicted))
    {
        countLinesInUpdatedFile++;
        printwriter.println(line);
    }

    //Hvis linje på form ".. UniRef100 entry.."
    if(!line.matches(uniref))
    {
        countLinesInUpdatedFile++;
        printwriter.println(line);
    }

    //Hvis linje på form "..hypothetical..");
    if(!line.matches(hypothetical))
    {
        countLinesInUpdatedFile++;
        printwriter.println(line);
    }
```java
// Hvis linje på form PREDICTED:....
if(!line.matches(predicted2))
{
    countLinesInUpdatedFile++;
    PrintWriter.println(line);
}

// Hvis 9.8
if(!line.matches(decimal))
{
    countLinesInUpdatedFile++;
    Proteintable.put(line, i);
    PrintWriter.println(line);
}

} /*
}
*/
br.close();
pri.close();

// legge til proteintable til dictionary-fila
System.out.println(proteintable.size());
// String protienames_to_dictionary = proteintable.toString();
// dictionaryPrintWriter.println(protienames_to_dictionary);
System.out.println("Antall linjer: " + countTotalLines);
System.out.println("Fjernede linjer: " + (countTotalLines - countLinesInUpdatedFile));
System.out.println("Prosent av opprinnelig fil: " + (double) countLinesInUpdatedFile / countTotalLines);
}

} catch(Exception e)
{
    e.printStackTrace();
}

/* Testmetode.
*/
private void testremoveEntries()
{
    try{
        int countTotalLines = 0;
        int countLinesWithoutNumbers = 0;
        String line;
        while( (line = br3.readLine()) != null )
        {
```
//den skriver ut annenhver linje tompa. \r-tesnet, derfor må teste lengde større enn 0
if(line.length() > 0)
{
    countTotalLines++;  
    if(line.matches("(H|h)(ypothetical).\*\))
    {
        System.out.println("treff");
    }
}
}
br3.close();
pw3.close();
System.out.println("Antall linjer: " + countTotalLines);
System.out.println("Fjernede linjer: " + (countTotalLines− countLinesWithoutNumbers));
System.out.println("Prosent av opprinnelig fil: "+ (double)countLinesWithoutNumbers/countTotalLines);
}  
}catch(Exception e)
{
e.printStackTrace();
}  
}  
}
package master;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/*/ 
 *Denne klassen skal foreta selve oppslaget i
 *protein og gen-dictionaries. Den får som input
 *Arraylisten finalList som inneholder ferdig
 *preprosesserte tokens fra n abstracts.
 *Den kaller Dictionary-klassen.
 *Den får også som input pmid-strengen, GUI-objektet, en peker
 *til PreProcessing, en ArrayList med de taggede proteinene,
 *en long-variabel som inneholder start-tiden
 *og arraylisten tokFreq som inneholder alle abstractene
 *sortert på nøkkel.
 */

public class LookUp {

    private String pmid;
    private PrintWriter pw;
    private ArrayList finalList, tokFreq;
    private ArrayList superGeneList, superProteinList, superList;
    private Dictionary d;
    private String token = "", tokenP1 = "", tokenP2 = "", tokenP3 = "", tokenP4 = "", tokenP5 = "", tokenP6 = "", tokenP7 = "", tokenP8 = "", tokenP9 = "";
    private GUI gui;
    private PreProcessing pp;
    private ArrayList relevantList;
    private ArrayList relevantBioTermsList;
    private ArrayList allBioTermHits;
    private ArrayList yapexTaggedProteins;
    private ArrayList sortedKeys, sortedFrequencies;
    private ArrayList postProcessingStopwords;
    private BufferedReader preader;
    private Hashtable ptable;

    public LookUp(String pmid, ArrayList finalList, GUI gui, ArrayList tokFreq, PreProcessing pp, ArrayList yapexTaggedProteins, long now)
```java
    
    { }
    
    this.pmid = pmid;
    this.finalList = finalList;
    this.gui = gui;
    this.pp = pp;
    this.tokFreq = tokFreq;
    this.yapexTaggedProteins = yapexTaggedProteins;
    superGeneList = new ArrayList();
    superProteinList = new ArrayList();
    superList = new ArrayList();
    allBioTermHits = new ArrayList();
    sortedKeys = new ArrayList();
    sortedFrequencies = new ArrayList();
    initializeStopwordsArray();
    d = new Dictionary(false);
    //printGoldenStandard();
    Weight w = new Weight();
    //beregner de tre mest relevante termer generelt
    ArrayList a = w.calculateWeight(tokFreq);
    relevantList = w.extractTopThreeMostRelevant(a);
    
    /*
     * Kommentert ut for å kjøre Test
     * 
     * insertRelevantTermsToGUI(relevantList);
     */
    
    //leser abstractene og returnerer bioterm-treffene
    ArrayList b = readAbstract(finalList);
    System.out.println("allBioTermHits.size before stopword-removal: " +
    allBioTermHits.size());
    
    //Fjerner stopord
    Stopwords stop = new Stopwords();
    allBioTermHits = stop.removeStopwords(allBioTermHits);
    System.out.println("allBioTermHits.size etter stopword-removal: " +
    allBioTermHits.size());
    removeBioTermEntries();
    System.out.println("allBioTermHits : Ny størrelse: "+allBioTermHits.size());
    sortBioTerms();
    System.out.println("allBioTermHits : Ny størrelse: "+allBioTermHits.size());
    
    ArrayList allBioTermHitsClone = (ArrayList)allBioTermHits.clone();
    ArrayList yapexTaggedProteinsClone = (ArrayList)yapexTaggedProteins.clone();
    
    //sammenligne de to arraylistene allBioTermHits og yapexTaggedProteins for å
    //finne recall og precision
    int answerSetSize = allBioTermHitsClone.size();
    int relevantSetSize = yapexTaggedProteinsClone.size();
    int measurementcounter = 0;
    
    //for all retrieved names
    for(int size = 0; size < allBioTermHits.size();size++)
    {
        
        //for all names in relevant set
        for(int u=0;u<yapexTaggedProteins.size();u++)
        
```
```java
if ( ((String)yapexTaggedProteins.get(u)).equals( (String)allBioTermHits.get(size) ) ) {
    measurementcounter++;
    allBioTermHits.remove(size);
    yapexTaggedProteins.remove(u);
    if (size != 0) {
        size--;
    }
    if (u != 0) {
        u--;
    }
}
}

double precision = (double)measurementcounter/answerSetSize;
double recall = (double)measurementcounter/relevantSetSize;

System.out.println("PRECISION: "+precision);
System.out.println("RECALL: "+ recall);

//sender superGeneList og superProteinList til sortering
ArrayList b0 = (ArrayList)b.get(0);
ArrayList b1 = (ArrayList)b.get(1);

/*
geneSortedList og protSortedList skal nå inneholde
sortedKeys, sortedFrequencies - AL-par
*/
ArrayList geneSortedList = pp.sortTokensByFrequency(b0);
ArrayList protSortedList = pp.sortTokensByFrequency(b1);

ArrayList resultList = new ArrayList();
resultList.add(geneSortedList);
resultList.add(protSortedList);
System.out.println("JENNIFER");

/*
*Kommenterte ut for å kjøre Test
*
*Result r = new Result(resultList, gui);
*/

ArrayList c = w.calculateWeight(geneSortedList);
ArrayList d = w.calculateWeight(protSortedList);

/*
*Kommentert ut for å kjøre Test
*
//beregner de fem mest relevante biotermer
relevantBioTermsList = w.calculateFiveMostRelevantBioTerms(c,d);
insertRelevantBiotermsToGUI(relevantBioTermsList);
*/

long after = System.currentTimeMillis();
System.out.println("Timeconsumption: "+(double)(after-now)/1000);
```

private void initializeStopwordsArray()
{
    BufferedReader br = null;
    try {
        br = new BufferedReader(new FileReader(new File("postProcessStopwords.txt")));
    }
    catch (Exception e) {}
    postProcessingStopwords = new ArrayList();
    try {
        String line = "";
        while ( (line = br.readLine()) != null )
        {
            if(line.length() > 0)
            {
                postProcessingStopwords.add(line);
            }
        }
    }
    catch (Exception e) {}
    System.out.println("Størrelse på stopwordsAL: "+postProcessingStopwords.size());
}

private void sortBioTerms()
{
    PrintWriter pri = null;
    try {
        pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("myBioTermsFrequency.txt")),false));
    }
    catch (Exception e) {}
    Hashtable ht = new Hashtable();
    Integer i = new Integer(1);
    for(int c = 0; c < allBioTermHits.size(); c++)
    {
        if(ht.containsKey( (String)allBioTermHits.get(c) ) )
        {
            Integer u = (Integer)ht.get((String)allBioTermHits.get(c));
            j+= u.intValue();
            u = new Integer(j);
            ht.put((String)allBioTermHits.get(c) , u);
        }
    }
}
else {
  ht.put((String)allBioTermHits.get(c),i);
}

System.out.println("Antall unike nøkler: "+ht.size());
Set s = ht.keySet();
Iterator it = s.iterator();
int n = 0;
int[] frequency = new int[ht.size()];
while(it.hasNext()) {
  String bio = ((String)it.next());
  Integer fr = (Integer)ht.get(bio);
  int a = fr.intValue();
  frequency[n] = a;
  n++;
}

try{
  sort(frequency);
} catch(Exception ex){ex.printStackTrace();}
Set newset = ht.keySet();
Iterator newit = newset.iterator();
Integer io = new Integer(0);
String xyz = "";
for(int xy = 0; xy < newset.size(); xy++) {
  sortedKeys.add(xyz);
  sortedFrequencies.add(io);
}
int test = 0;
boolean found = false;
// For alle mappinger i HashMap tokensCounted
while(newit.hasNext()) {
  test++;
  found = false;
  String m = ((String)newit.next());
  Integer e = (Integer)ht.get(m);
  int a = e.intValue();
  if(!found){
    // Itererer over frequency-arrayet
    for(int k = 0; k < frequency.length;k++) {
      if(!found){
        // Hvis verdi a fra tokensCounted er lik verdi i frequency-array
        if(a == frequency[k]) {
          sortedKeys.set(k,m);
          sortedFrequencies.set(k,e);
          frequency[k]=0;
          found = true;
        }
      }
    }
    found = true;
  }
}
for (int y = 0; y < sortedKeys.size(); y++)
{
    pri.println((String)sortedKeys.get(y) + " " + "," + " " + (Integer) 
sortedFrequencies.get(y));
}

pri.close();

private void printGoldenStandard()
{
    PrintWriter printx = null;
    try{
        printx = new PrintWriter(new BufferedWriter(new FileWriter(
new File("goldenstandard.txt"), false)));
    }catch(IOException io){}

    for (int e = 0; e < yapexTaggedProteins.size(); e++)
    {
        printx.println((String)yapexTaggedProteins.get(e));
    }
    printx.close();
}

private void printMyBioTermHits()
{
    /*
     * Her inneholder nå allBioTermHits alle protein/gen-treff
     * som systemet fant gjennom test på Yapex-teksten.
     * Sammenligner de med yapexTaggedProteins for å få målinger
     * på recall og precision.
     */
    PrintWriter printy = null;
    try{
        printy = new PrintWriter(new BufferedWriter(new FileWriter(
new File("myBioTerms.txt"), false)));
    }catch(IOException io){}

    for (int e = 0; e < allBioTermHits.size(); e++)
    {
        printy.println((String)allBioTermHits.get(e));
    }
    printy.close();
}

/*
 * Denne metoden skaper en one-level hash table
 * av proteinnavnene. Her må en hardkode dersom
 * man ønsker at systemet skal benytte stemming
 * istedet for ikke stemming. Denne one-level
 * hash tabellen brukes for å fjerne irrelevante
 * treff av lengde 1 ord
 */
private void pHT()
{
    try{
        ClassLoader cl = this.getClass().getClassLoader();
        URL pURL = cl.getResource("final_dictionaries/proteinDictionary2.txt");
        URLConnection pConnection = pURL.openConnection();
        pConnection.setDoInput(true);
        pConnection.connect();
        preader = new BufferedReader(new InputStreamReader(pConnection.getInputStream()));
    }catch(Exception e){}
    ptable = new Hashtable();
    Integer i = new Integer(0);
    String line = "";
    try{
        while( (line = preader.readLine()) != null )
            if(line.length() > 0)
                ptable.put(line, i);
    }
    System.out.println("One-level protein hashtable size: "+ptable.size());
}
}
private void removeBioTermEntries()
{
    pHT();
    int number = 0;
    int number1 = 0;
    int number2 = 0;
    int number3 = 0;
    int number4 = 0;
    int number5 = 0;
    Pattern p = Pattern.compile(\"\d\\); //rene tallinnslag
    Pattern p1 = Pattern.compile(\"D\(1,2\)\); //et ikke-tall-innslag 1 til 2 tegn
       langt
    Pattern p2 = Pattern.compile(\"d*\(\u002E)\d*\); //desimaltall
    Pattern p3 = Pattern.compile(\"d*\s(kda)\(\s((P|p)rotein)\)\.*\); //kda
       innslag
    for(int v = 0; v<allBioTermHits.size();v++)
    {
        String bioTerm = (String)allBioTermHits.get(v);
        String[]t = bioTerm.split("\s");
        int numberoftokens = t.length;
        Matcher m = p.matcher(bioTerm);
        Matcher m1 = p1.matcher(bioTerm);
        Matcher m2 = p2.matcher(bioTerm);
    }
Matcher m3 = p3.matcher(bioTerm);

if(m.matches())
{
    number++;
    allBioTermHits.remove(v);
    v--;
}
else if(m1.matches())
{
    number1++;
    allBioTermHits.remove(v);
    v--;
}
else if(m2.matches())
{
    number3++;
    allBioTermHits.remove(v);
    v--;
}
else if(m3.matches())
{
    number5++;
    allBioTermHits.remove(v);
    v--;
}
else if(numberoftokens==1 && !ptable.containsKey(bioTerm))
{
    number4++;
    allBioTermHits.remove(v);
    v--;
}
else{
    for(int q = 0; q < postProcessingStopwords.size();q++)
    {
        if(bioTerm.equals((String)postProcessingStopwords.get(q)))
        {
            number2++;
            allBioTermHits.remove(v);
            v--;
        }
    }
}
System.out.println("Antall rene tallinnslag fjernet: "+number);
System.out.println("Antall 1 til 2 tegn lange ord fjernet: "+number1);
System.out.println("Antall ord som matcher postprocstopwords og er fjernet: "+number2);
System.out.println("Antall desimaltall fjernet: "+number3);
System.out.println("Antall 1-ord lange innslag som ikke er selvstendige: "+number4);
System.out.println("Antall kda-innslag fjernet: "+number5);
printMyBioTermHits();
}

private void insertRelevantBioTermsToGUI(ArrayList relevantBioTermsList)
String one = (String)relevantBioTermsList.get(0);
String two = (String)relevantBioTermsList.get(1);
String three = (String)relevantBioTermsList.get(2);
String four = (String)relevantBioTermsList.get(3);
String five = (String)relevantBioTermsList.get(4);

gui.insertTopFiveRelevantBioTerms(one,0);
gui.insertTopFiveRelevantBioTerms(two,1);
gui.insertTopFiveRelevantBioTerms(three,2);
gui.insertTopFiveRelevantBioTerms(four,3);
gui.insertTopFiveRelevantBioTerms(five,4);
}

public ArrayList readAbstract(ArrayList finalList) {
  String s = " ";
  int countGeneHits=0;
  int countProteinHits=0;

  int length2 = 0;
  int length3 = 0;
  int length4 = 0;
  int length5 = 0;
  int length6 = 0;
  int length7 = 0;
  int length8 = 0;
  int length9 = 0;
  int length10 = 0;

  boolean outOfIndex = false;
  ArrayList geneHits = new ArrayList();
  ArrayList proteinHits = new ArrayList();

  System.out.println("finalList.size: " + finalList.size());
  for(int q = 0; q < finalList.size(); q++) {
    int geneHit = 0;
    ...
int proteinHit = 0;
int bioTermHit = 0;

ArrayList abstractText = (ArrayList) finalList.get(q);

for (int a = 0; a < abstractText.size(); a++)
{
    outOfIndex = false;
    token = (String) abstractText.get(a);
    if (a < abstractText.size() - 1) { tokenP1 = (String) abstractText.get(a + 1); }
    else { outOfIndex = true; }
    if (a < abstractText.size() - 2) { tokenP2 = (String) abstractText.get(a + 2); }
    else { outOfIndex = true; }
    if (a < abstractText.size() - 3) { tokenP3 = (String) abstractText.get(a + 3); }
    else { outOfIndex = true; }
    if (a < abstractText.size() - 4) { tokenP4 = (String) abstractText.get(a + 4); }
    else { outOfIndex = true; }
    if (a < abstractText.size() - 5) { tokenP5 = (String) abstractText.get(a + 5); }
    else { outOfIndex = true; }
    if (a < abstractText.size() - 6) { tokenP6 = (String) abstractText.get(a + 6); }
    else { outOfIndex = true; }
    if (a < abstractText.size() - 7) { tokenP7 = (String) abstractText.get(a + 7); }
    else { outOfIndex = true; }
    if (a < abstractText.size() - 8) { tokenP8 = (String) abstractText.get(a + 8); }
    else { outOfIndex = true; }
    if (a < abstractText.size() - 9) { tokenP9 = (String) abstractText.get(a + 9); }
    else { outOfIndex = true; }

    boolean gene = performGeneDictionaryLookup(token);
    boolean protein = performProteinDictionaryLookup(token, 0);
    // Hvis gennavn, adder til gen-navnlister
    if (gene)
    {
        geneHit++;
        bioTermHit++;
        geneHits.add(token);
        // allBioTermHits er laget for å kjøre Test
        allBioTermHits.add(token);
        // Hvis proteinnavn, adder til protein-navnlister
        // undersøk om neste ord er på neste nivå i dictionary osv..
        else if (protein)
        {
            String tokenString = "";
            proteinHits.add(proteinHit, token);
            tokenString = token;
            allBioTermHits.add(bioTermHit, tokenString);
            // 2. ord i navnet
            if (!outOfIndex && performProteinDictionaryLookup(tokenP1, 1))
            { }
//System.out.println("Fått treff på andre nivå");
length2++;
proteinHits.remove(proteinHit);
proteinHits.add(proteinHit, token+s+tokenP1);
a++;
tokenString = token+s+tokenP1;
allBioTermHits.remove(bioTermHit);
allBioTermHits.add(bioTermHit,tokenString);

//3. ord
if( !outOfIndex && performProteinDictionaryLookup(tokenP2,2))
{
    //System.out.println("Fått treff på tredje nivå");
    length3++;
    proteinHits.remove(proteinHit);
    proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2);
a++;
tokenString = token+s+tokenP1+s+tokenP2;
allBioTermHits.remove(bioTermHit);
allBioTermHits.add(bioTermHit,tokenString);
}

//4. ord
if( !outOfIndex && performProteinDictionaryLookup(tokenP3,3))
{
    //System.out.println("Fått treff på fjerde nivå");
    length4++;
    proteinHits.remove(proteinHit);
    proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+tokenP3);
a++;
tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3;
    //System.out.println("4. nivå: "+tokenString);
allBioTermHits.remove(bioTermHit);
allBioTermHits.add(bioTermHit,tokenString);
}

//5. ord
if( !outOfIndex && performProteinDictionaryLookup(tokenP4,4))
{
    length5++;
    proteinHits.remove(proteinHit);
    proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4);
a++;
tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4;
allBioTermHits.remove(bioTermHit);
allBioTermHits.add(bioTermHit,tokenString);
}

//6. ord
if( !outOfIndex && performProteinDictionaryLookup(tokenP5,5))
{
    length6++;
    proteinHits.remove(proteinHit);
    proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5);
a++;
tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5;
allBioTermHits.remove(bioTermHit);
allBioTermHits.add(bioTermHit, tokenString);

// 7.ord
if ( !outOfIndex && performProteinDictionaryLookup(tokenP6, 6)) {
    length7++;
    proteinHits.remove(proteinHit);
    proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5+s+tokenP6);
    a++;
    tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5+s+tokenP6;
    allBioTermHits.remove(bioTermHit);
    allBioTermHits.add(bioTermHit, tokenString);
}

// 8.ord
if ( !outOfIndex && performProteinDictionaryLookup(tokenP7, 7)) {
    length8++;
    proteinHits.remove(proteinHit);
    proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5+s+tokenP6+s+tokenP7);
    a++;
    tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5+s+tokenP6+s+tokenP7;
    allBioTermHits.remove(bioTermHit);
    allBioTermHits.add(bioTermHit, tokenString);
}

// 9.ord
if ( !outOfIndex && performProteinDictionaryLookup(tokenP8, 8)) {
    length9++;
    proteinHits.remove(proteinHit);
    proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5+s+tokenP6+s+tokenP7+s+tokenP8);
    a++;
    tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5+s+tokenP6+s+tokenP7+s+tokenP8;
    allBioTermHits.remove(bioTermHit);
    allBioTermHits.add(bioTermHit, tokenString);
}

// 10.ord
if ( !outOfIndex && performProteinDictionaryLookup(tokenP9, 9)) {
    length10++;
    proteinHits.remove(proteinHit);
    proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5+s+tokenP6+s+tokenP7+s+tokenP8+s+tokenP9);
    a++;
    tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s+tokenP5+s+tokenP6+s+tokenP7+s+tokenP8+s+tokenP9;
    allBioTermHits.remove(bioTermHit);
}
allBioTermHits.add(bioTermHit, tokenString);
}
}
}
}
}
}
proteinHit++;
bioTermHit++;
}
// Ordet er uinteressant
else {

geneHitsClone = (ArrayList)geneHits.clone();
proteinHitsClone = (ArrayList)proteinHits.clone();
countGeneHits = countGeneHits+geneHitsClone.size();
countProteinHits = countProteinHits+proteinHitsClone.size();
superGeneList.add(geneHitsClone);
superProteinList.add(proteinHitsClone);
geneHits.clear();
proteinHits.clear();
}
superList.add(geneHitsClone);
superList.add(proteinHitsClone);
return superList;

/*
* Denne metoden foretar Lookup i protein-dictionary.
* Returnerer en boolean som forteller om det
* aktuelle token er del av et proteinnavn eller ikke.
*/
public boolean performProteinDictionaryLookup(String possibleProtein, int hashTableLevel)
{
    boolean protein = false;
    switch(hashTableLevel)
    {
    case 0: protein = d.proteinDictionary_level_0.containsKey(possibleProtein);
            break;
    case 1: Hashtable intern0 = (Hashtable)d.proteinDictionary_level_0.get(token);
            protein = intern0.containsKey(possibleProtein);
            break;
    case 2: Hashtable intern1 = (Hashtable)d.proteinDictionary_level_1.get(tokenP1);
            protein = intern1.containsKey(possibleProtein);
            break;
    case 3: Hashtable intern2 = (Hashtable)d.proteinDictionary_level_2.get(tokenP2);
            protein = intern2.containsKey(possibleProtein);
            break;
    case 4 : Hashtable intern3 = (Hashtable)d.proteinDictionary_level_3.get(tokenP3);
            protein = intern3.containsKey(possibleProtein);
            break;
    case 5: Hashtable intern4 = (Hashtable)d.proteinDictionary_level_4.get(tokenP4);
            protein = intern4.containsKey(possibleProtein);
            break;
    case 6: Hashtable intern5 = (Hashtable)d.proteinDictionary_level_5.get(tokenP5);
            protein = intern5.containsKey(possibleProtein);
            break;
    case 7: Hashtable intern6 = (Hashtable)d.proteinDictionary_level_6.get(tokenP6);
            protein = intern6.containsKey(possibleProtein);
            break;
    case 8:Hashtable intern7 = (Hashtable)d.proteinDictionary_level_7.get(tokenP7);
            protein = intern7.containsKey(possibleProtein);
            break;
    case 9: Hashtable intern8 = (Hashtable)d.proteinDictionary_level_8.get(tokenP8);
            protein = intern8.containsKey(possibleProtein);
            break;
    default : protein = false;
            break;
    }
    return protein;
}
Debe metoden foretar lookup i gen-dictionary.
*Returnerer en boolean som indikerer om tokenet er et gen.*

```
public boolean performGeneDictionaryLookup(String possibleGene)
{
    //System.out.println("geneDictionary er lik null: "+d.geneDictionary.equals(null));
    boolean gene = d.geneDictionary.containsKey(possibleGene);
    return gene;
}
```

Metodene nedenfor er del av quicksort-algoritmen.

```
private void quickSort(int a[], int l, int r) throws Exception
{
    int M = 4;
    int i;
    int j;
    int v;
    if ((r-l)>M)
    {
        i = (r+l)/2;
        if (a[l]>a[i]) swap(a,l,i);  // Tri-Median Methode!
        if (a[l]>a[r]) swap(a,l,r);
        if (a[i]>a[r]) swap(a,i,r);
        j = r-1;
        swap(a,i,j);
        i = l;
        v = a[j];
        for(;;)
        {
            while(a[++i]<v);
            while(a[--j]>v);
            if (j<i) break;
            swap(a,i,j);
        }
        swap(a,i,r-1);
        quickSort(a,l,j);
        quickSort(a,i+1,r);
    }
    else
    {
        swap(a,l,r-1);
        quickSort(a,l,j);
        quickSort(a,i+1,r);
    }

    private void swap(int a[], int i, int j)
    {
        int T;
        T = a[i];
        a[i] = a[j];
        a[j] = T;
    }
```

```
private void insertionSort(int a[], int l00, int h00) throws Exception
{
    int i;
    int j;
    int v;
```
for (i=lo0+1;i<=hi0;i++)
{
    v = a[i];
    j=i;
    while ((j>lo0) && (a[j-1]>v))
    {
        a[j] = a[j-1];
        j--;
    }
    a[j] = v;
}

public void sort(int a[]) throws Exception
{
    quickSort(a, 0, a.length - 1);
    insertionSort(a, 0, a.length-1);
}
References


[6] Analysis of Protein/Protein Interactions Through Biomedical Literature: Text Mining of Abstracts vs. Text Mining of Full Text Articles, Martin et al., 2004


[12] Protein names and how to find them, Franzén et al., Stockholm, 2002


[19] *What is the Document Object Model?*, [http://www.w3.org/TR/DOM-Level-3-Core/introduction.html](http://www.w3.org/TR/DOM-Level-3-Core/introduction.html)

[20] *XML DOM Introduction*, [http://www.w3schools.com/dom](http://www.w3schools.com/dom)


[23] *Analysis of Biomedical Text for Chemical Names: A Comparison of Three Methods*, W.John Wilbur et al., National Center for Biotechnology Information (NCBI)


[27] *Semantic Annotation of Biomedical Literature using Google*, Saetre et al., NTNU, May 2005