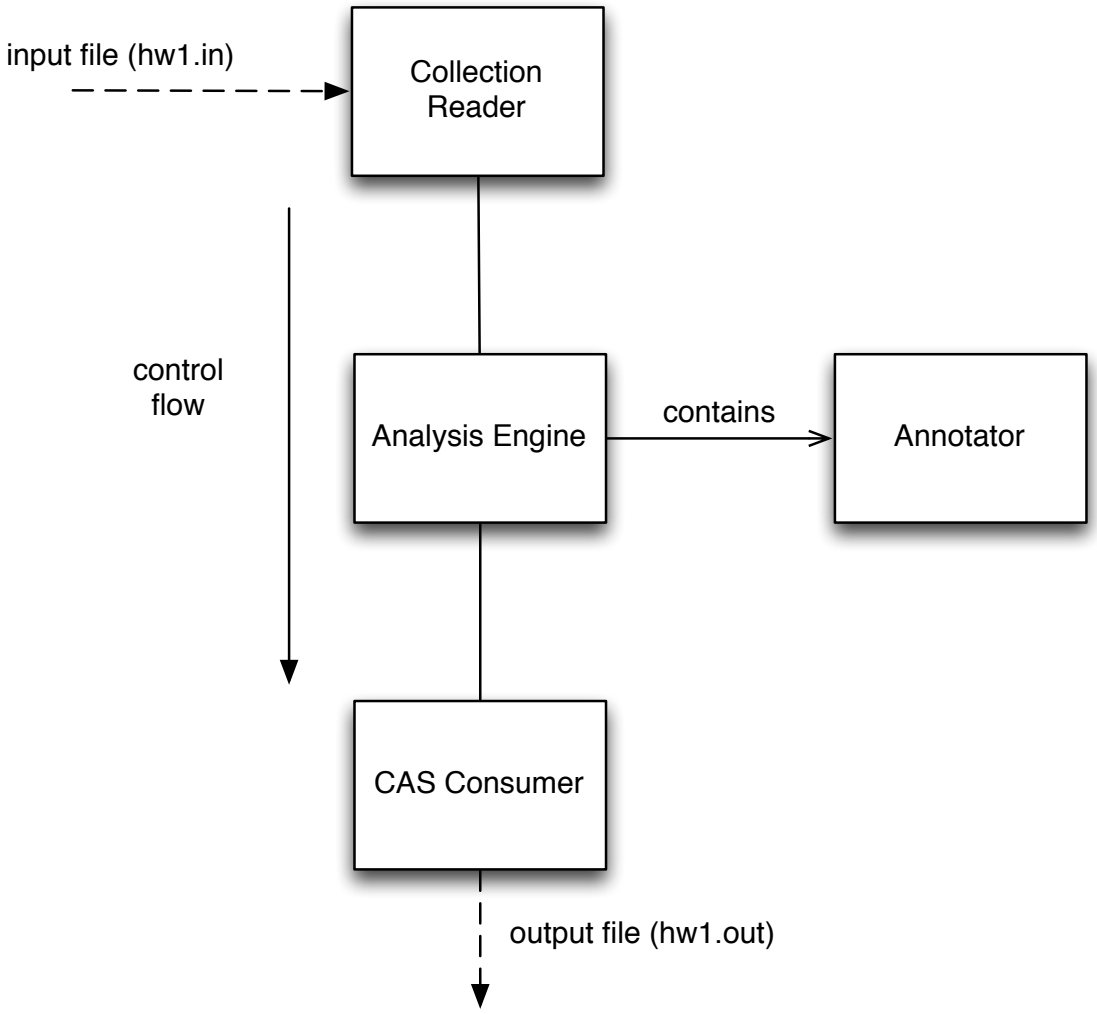
Report of Homework 1

Student Name: Yuliang Yin
Andrew ID: yuliangy

Part 1. General Structure and Design

I. Overall Structure Description

The whole Named-Entity Recognition application is based on the CPE pipeline of UIMA framework. The structure includes three components basically: collection reader, analysis engine and CAS consumer. Specifically, analysis engine includes an annotator that is responsible for the main logic stuffs. A high-level structure diagram is shown below.



Here, the control flow is sequentially from collection reader to analysis engine (annotator) and to CAS consumer.

II. Detailed Design of System

In this section, I would like to describe some detailed design of the system with regard to each component.

Collection Reader

My collection reader is simple and straightforward. It simply loads the document into the CAS. In order to find the document, I define a parameter named "InputFile" which indicates the relative path of the input file in the reader descriptor. It is quite similar to the "FileCollectionReader" in the UIMA examples. The only difference is that my collection reader only reads a single file.

Even though the implementation is simple, it is still based on some tradeoff. We can find that such implementation is not good at dealing with the document whose size is on the level of GB since the memory would be insufficient. But if we try to let the reader load one sentence each time, it will creates overhead and is hard for us to use UIMA Document analyzer to visualize the result if your annotator is very specific to one sentence. Since the sample input is not that big, a simple and straightforward implementation is efficient.

Analysis Engine

My analysis engine only contains a single annotator that is named "GeneMentionAnnotator". In the process function of the annotator, I implement the main logic of the application that can be divided into three steps generally.

a. Read the model from external source. In this application, I use the **LingPipe** package to help me identify the gene tags in a document. Before we create a chunker (the recognizer defined in LingPipe), we need to load the trained model file (I use **ne-en-bio-genetag.HmmChunker** in this case) first. In order to load the external source, I bind the resource in the annotator descriptor.

b. Scan the text and find gene tags using HMM chunker. This step is easy to implement. After chunking each sentence, we get a set of chunks.

c. Add annotation to CAS. For each chunk, I extract the *begin* and *end* information. And based on the confidence (whether >= 0.2), I decide whether to consider a chunk as a true gene tag.

   The annotator outputs a GeneMention type that I define in "GeneMentionDescriptor". Besides the *begin* and *end* inherited from "Annotation", GeneMention also contains *SentenceIdentifier* *StartOffset*, *EndOffset*, and Phrase. *StartOffset* and *EndOffset* is the

location of the phrase ignoring the white spaces. I do not set *begin* and *end* to those offset values because *begin* and *end* features can be used for UIMA document analyzer for visualization.

CAS Consumer

In my system, the CAS consumer is simply a modified version of "AnnotationPrinter" provided by UIMA examples. I just make the modification in the output content part so that it will output correct things. Also, an OutputFile path parameter is set in the descriptor.

Part 2. Evaluation and Analysis

I. Evaluation

I write a simple program named "Estimator" to evaluate the result of the NER system compared to "sample.out" provided.

Under the assumption that "sample.out" contains a complete and correct set of gene tags, I measure the precision, recall and F1 score for the result and regarding to different confidence threshold. The evaluation results are shown in the table below.

Confidence Threshold	Precision	Recall	F1 score
0.8	0.8724	0.7403	0.8009
0.5	0.7872	0.8314	0.8087
0.2	0.6723	0.8879	0.7653
0.05	0.5426	0.9240	0.6837

Based on this table, we can find that obviously precision and recall follow opposite trends while F1 score has an optimum. Therefore, I choose to set the threshold to be 0.5.

II. Preliminary Analysis

a. We can easily find that the result from the LingPipe is quite similar to the "sample.out", which indicates that such method is reliable and effective. But it still needs improvement since there are still some gene tags are not recognized and some gene tags are considered useless by "sample.out". This bias between the results is due to the difference between models and some assumptions. A better way to solve this is to train a filter based on "sample.out" to perform on the results of the LingPipe so that the gene tags identified would be of higher confidence with the combined effects of "sample.out" and LingPipe.

b. Moreover, my design is straightforward and with lower complexity which makes it easier to understand. Also, such design is based on some assumptions. Firstly, I assume that the document is not that big and can be loaded to the memory easily. Secondly, it is easy for us to generate an output file and also use document analyzer to visualize it.

   But if the assumptions are not satisfied, it is still adaptive to some changes. For instance, I can modify the reader and the consumer to do one sentence process which will creates some overhead and also make the reader and consumer more coupled since file I/O process should be more consistent (e.g. maintaining some global information).