11-791 Design and Engineering of Intelligent Information System Fall 2012 Homework 1

UML Design and Named Entity Recognition Implementation with UIMA SDK

Important dates

- **Hand out: October 2.**[a]
- **Turn in: October 16.** Besides all your Java source codes, and UIMA type system, collection reader and analysis engine descriptors, which are essential to your UIMA collection processing engine, you are also required to answer UML design related questions, write a report for your implementation of the named entity recognizer, and include the Javadocs.

  If you have ever looked into your `target/` directory or Maven course-release repository for Homework 0, then you will find that Maven will package the binary files, source files, and Javadocs into different jars, and deploy them on the server, which means all you need to do is to put all the source codes and descriptors, as well as the documents, in the right place of your hw1-ID project. You should organize your project in the same hierarchy as shown below[b]:

```
hw1-ID
|- pom.xml
`- src
   `- main
      |- java
      |  `- **/*.java            /* your Java codes go into this folder
      |                           * as you did before */
      `- resources
         |- CpeDescriptor.xml    /* the entry point for your cpe */
         |- **/*.xml             /* all your descriptors go into the
         |                        * resources folder */
         `- docs
            |- hw1-ID-uml.pdf    /* your UML design writeup */
            `- hw1-ID-report.pdf /* your report for the pipeline */
```

[a] This version was built on September 28, 2012

[b] To simplify your submission process and our evaluation process, we ask you to create `src/main/resources/docs` for your documents. But you should keep in mind it is NOT a good practise to have documentation within a `src` folder.

Several notes about organizing your Maven project and other additional information:

1. **Submission:** The same way as you did for Homework 0 (set up GitHub repo, create Maven project, write your code, submit to Maven repo), except that the name has changed to hw1-ID.

2. **Your source files and descriptors:** `**/*.java` and `**/*.xml` tell you that you don't need to flatten your folder hierarchy, instead we encourage you to place your Java codes in the right package, and similarly, you can create subfolders for different types of descriptors, e.g., `src\main\resources\descriptors\ner` for all the analysis engine descriptors related to named entity recognition task. We will look into your jar packages.

   One very special descriptor is your `CpeDescriptor.xml`, where is the entry point for the entire pipeline, which means you are required to name your CPE descriptor exactly the same way, and place it in the right place for us to evaluate your code.

3. **UML design answers and report:** We will pull out your documents from your jar files, and Prof. Eric Nyberg will look into your report and answers for the questions, so that remember to include your ID as part of file names, and put your name and ID in your document. You can submit either PDF files or DOC/DOCX files.

4. **Javadocs:** Please refer to any best practise (yes, there might be more than one) to write your Javadoc, e.g., `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`. We expect you to describe your major components at the class level of your Javadoc, and put additional comments on the most important methods if any. It might be a good idea to include your UML design in your Javadoc if you think the developers can have a general idea by just looking at your diagram.

5. **Performance evaluation:** We believe you may have different ideas about how to design the type system, and how to implement a name entity recognizer, which means we won't have a shared type system to test your components (but we will have one in Homework 2). Therefore, you are required to write a collection reader to load the input text and a CAS consumer to generate your output (both are independent of your specific type system), and our evaluation will based on the output from your CAS consumer, similar as we did for Homework 0.

Useful information

1. We encourage you to start Homework 1 as early as possible, especially if you haven't gotten a chance to browse the UIMA portal.

2. For any question regarding the UML design part of Homework 1, please send mails directly to Prof. Eric Nyberg (`ehn@cs.cmu.edu`), and for other questions, please send us mails: Zi Yang (`ziy@cs.cmu.edu`) or Rui Liu (`ruil@cs.cmu.edu`).

3. Again, both source files and pdf file of this assignment are publicly available on my GitHub

   `http://github.com/ziy/software-engineering-preliminary`

   Please feel free to fork the project and send a pull request back to me as some of you did for Homework 0 for any error. Or you can just report an issue at

   `http://github.com/ziy/software-engineering-preliminary/issues`

Task 1

UML Design

In this task, you are required to answer some questions about UML design. Once you've answered all the questions, remembered to convert whatever format your document is to PDF, and name it as hw1-ID-report.pdf.

Task 1.1 Domain diagram for IntelligentInformationSystem

"An IntelligentInformationSystem is composed of a sequence of data processing operations or phases. Each phase accepts certain data types as input and produces certain data types as output. Each phase can be implemented by any number of algorithms or options. Each option is implemented by a specific Java class. Each option may have any number of configuration parameters; each configuration parameter has some set of acceptable values."

Draw a UML Domain Diagram to represent the domain concepts, associations (with multiplicities) and attributes expressed in the description above.

Task 1.2 Domain diagram for AnalysisEngine

"An AnalysisEngine is composed of a sequence of algorithms or options. Each option accepts certain data types as input and produces certain data types as output. Each option is implemented by a specific Java class. Each option has some number of configuration parameters; each configuration parameter has a specific assigned value."

Draw a UML Domain Diagram to represent the domain concepts, associations (with multiplicities) and attributes expressed in the description above.

Task 1.3 Sequence Diagram

There is a one-to-many relationship between IntelligentInformationSystem and AnalysisEngine. Assume that an IntelligentInformationSystem has the responsibility to produce a set of AnalysisEngines that represent all of the possible data flows in the IntelligentInformationSystem. Design a method with this signature:

```
ArrayList<CollectionProcessingEngine> instantiateEngines (
                          IntelligentInformationSystem iis);
```

Draw a UML Sequence Diagram to show the sequence of messages required to a) read the information from the IntelligentInformationSystem instance, b) instantiate the corresponding AnalysisEngine instances, and c) store the AnalysisEngine instances in a List, which is the final output of the program. The message and return value for this use case are illustrated below.

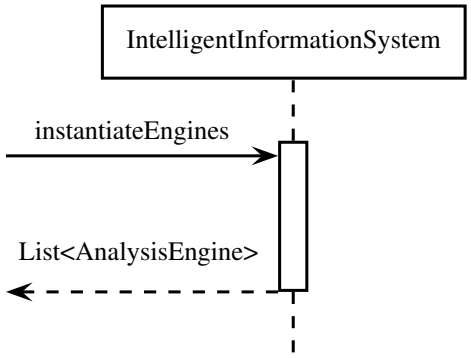

Figure 1.1: The message and return value for this use case

Finally, don't forget to put your name and Andrwe ID at the top of the document, name the file as "hw1-ID-uml.pdf" and put it under `src/main/resources/docs`.

Task 2

Knowing what a named entity is and what a named entity recognizer is

In this task, you will first learn what a named entity is and what a named entity recognizer is. Plus, we will give you clear definitions about what kinds of named entities you are required to extract from the text.

Task 2.1 Knowing basic concepts

From Wikipedia[1], the process of named entity recognition is defined as:

> Named-entity recognition (NER) is a subtask of information extraction that seeks to locate and classify atomic elements in text into predefined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

Several things you should be aware:

1. NER involves two subtasks: "locating" and "classifying", which correspond to the identification of the text span (i.e. begin and end) and the assignment of a label to the text respectively, which means NER task fits into the UIMA framework.
2. Named entities correspond to the atomic elements in text, which are not necessarily single tokens (those you finally print out to the console from Homework 0). Many times, named entities can correspond to several consecutive tokens, and many tokens may not be part of named entities.
3. NER plays a very crucial role in not only information extraction, but in all the text analysis related task, e.g. information retrieval and question answering. In the next homework, you will find NER can help you better identify the key words in the query and documents, and thus retrieve synonyms more accurately.

[1] `http://en.wikipedia.org/wiki/Named-entity_recognition`

Task 2.2 Learning gene mention tagging

For this task, you are going to tackle a specific NER problem in the biological domain, *Gene Mention Tagging*[2], and you are going to use partial labeled dataset from BioCreAtIvE (Critical Assessment of Information Extraction systems in Biology challenge evaluation)[3], which focuses on evaluating text mining and information extraction systems applied to the biological domain. Gene Mention Tagging task is concerned with the named entity extraction of gene and gene product mentions in text. For example, given a sentence from a biological literature: *Comparison with alkaline phosphatases and 5-nucleotidase*, you should identify *alkaline phosphatases* and *5-nucleotidase* correspond to gene names.

The input file will consist of ascii sentences, one per line. Each sentence will be preceded on the same line by a sentence identifier, i.e.,

```
sentence-identifier-1 text
```

For example, the sentence in the above example will be given to you as follows,

```
P00001606T0076 Comparison with alkaline phosphatases and 5-nucleotidase
```

You are given an example input file (consisting of 15,000 sentences, each per line, all from biological literature). We will later test your pipeline with a different dataset.

You are required to output an ascii list of reported gene name mentions, one per line, and formatted as:

```
sentence-identifier-1|start-offset-1 end-offset-1|optional text...
sentence-identifier-1|start-offset-2 end-offset-2|optional text...
sentence-identifier-1|start-offset-3 end-offset-3|optional text...
sentence-identifier-2|start-offset-1 end-offset-1|optional text...
sentence-identifier-3|start-offset-1 end-offset-1|optional text...
.
.
.
```

For example, we expect you generate the following to lines for the example sentence above:

```
P00001606T0076|14 33|alkaline phosphatases
P00001606T0076|37 50|5-nucleotidase
```

The sentence-identifier is from the sentence of the mention. Multiple mentions from the same sentence should appear on separate lines. A sentence is not required to have any mentions. The start-offset is the number of non-whitespace characters in the sentence preceding the first character of the mention, and the end-offset is the number of non-whitespace characters in the sentence preceding the last character of the mention. A sample output file will also be given to you, which contains all the identified gene mentions for the input file.

[2] http://www.biocreative.org/tasks/biocreative-ii/task-1a-gene-mention-tagging/

[3] `http://www.biocreative.org`

Task 2.3 Sketching your NER pipeline

We assume you are clear about the task, the input format, and the output, and now it's time to sketch your collection processing engine. How do you plan to design the type system? How do you plan to deal with input file and generate output file? What approach(es) are you going to use to identify the named entities? Do you also consider to incorporate any additional resources?

In the next task, you will practise to solve the problem within the UIMA framework.

Q1 I still have no idea about how named entities can be extracted from text? Could you share me some ideas?

A1 Named entity recognition is a long-studied problem in NLP, which means you can easily find many influential papers to propose novel ideas to tackle this issue every year, including specific to the biological domain. For example,

1. You can try to look for lexical/grammatical heuristics for the gene mentions, and summarize them in a rule-based algorithm. For example, many people find part-of-speech is very useful information to identity not only the named entities but also the text spans that cannot be named entities.
2. If you are familiar with machine learning, then you could try to use a sequence model like HMM or CRF to predict the state of each token, based on the features you might extract.
3. If you are familiar with biological knowledge resources, esp. gene name resources, you can definitely write a wrapper for the resource you think are relevant to the task, and look up candidate named entities via your wrapper.
4. etc.

We encourage you to try some novel ideas for your NER. You can incorporate an interesting feature in your learning framework, or you can try to combine multiple approaches in a more principled way. We know that each of you has a different background. So don't worry if you don't know how to propose a fancy sophisticated machine learning algorithm for this task.

Q2 Is there any restriction to use the sample input file? Can I use it as part of my training file for NER?

A2 Yes. You can do that. But keep in mind that we will use a different input file to evaluate your pipeline, so don't overfit your model! In addition, we encourage you to test your pipeline with the given input file to make sure your collection reader works properly, and test your pipeline with the sample output, and report the preliminary result in your report.

Q3 How were the gene mentions (gold-standard annotations) identified in the output file?

A3 According to the *GENETAG Annotation Guidelines*, there are several rules for the annotators to follow in order to determine which words are considered as part of a single gene/protein mention. If you are really curious about the details, you can refer to the Appendix for the guideline.

We show you a very simple named entity recognizer based on token's part-of-speech (e.g., noun, verb, adjective, adverb, etc.) in Listing 2.1. The idea of this named entity recognizer is simple. It first uses a tokenizer to tokenize the sentence into tokens, and then label each token with a part-of-speech tag. Finally, the consecutive nouns will be grouped into "noun phrases", which will be added to the list of name entities.

You can test the perform of this named entity recognizer with the collection processing engine that you will implement later. The output for the first sentence in the collection (i.e. the example sentence we used earlier for this task) is:

```
P00001606T0076|0 10|Comparison
P00001606T0076|16 37|alkaline phosphatases
P00001606T0076|42 56|5-nucleotidase
```

Luckily, we correctly identify two gene mentions with this simple approach. But "Comparison" is definitely not a gene mention. If you like to apply a similar approach, you could try to find out solutions to fix this. Got some ideas? I suggest you to write them down now since you need some additional steps before you can actually implement the components.

```
import java.util.*;

import edu.stanford.nlp.ling.CoreAnnotations.*;
import edu.stanford.nlp.pipeline.*;
import edu.stanford.nlp.util.*;

public class PosTagNamedEntityRecognizer {

  private StanfordCoreNLP pipeline;

  public PosTagNamedEntityRecognizer() {
    Properties props = new Properties();
    props.put("annotators", "tokenize, ssplit, pos");
    pipeline = new StanfordCoreNLP(props);
  }

  public Map<Integer, Integer> getGeneSpans(String text) {
    Map<Integer, Integer> begin2end = new HashMap<Integer, Integer>();
    Annotation document = new Annotation(text);
    pipeline.annotate(document);
    List<CoreMap> sentences = document.get(SentencesAnnotation.class);
    for (CoreMap sentence : sentences) {
      List<CoreLabel> candidate = new ArrayList<CoreLabel>();
      for (CoreLabel token : sentence.get(TokensAnnotation.class)) {
        String pos = token.get(PartOfSpeechAnnotation.class);
        if (pos.startsWith("NN")) {
          candidate.add(token);
        } else if (candidate.size() > 0) {
          int begin = candidate.get(0).beginPosition();
          int end = candidate.get(candidate.size() - 1).endPosition();
          begin2end.put(begin, end);
          candidate.clear();
        }
      }
      if (candidate.size() > 0) {
        int begin = candidate.get(0).beginPosition();
        int end = candidate.get(candidate.size() - 1).endPosition();
        begin2end.put(begin, end);
        candidate.clear();
      }
    }
    return begin2end;
  }
}
```

Listing 2.1: A simple name entity recognition program based on Stanford CoreNLP tool

Task 3

Implementing A Named Entity Recognizer with UIMA SDK

In this task, you need to implement a named entity recognizer based on UIMA SDK. We assume you have finshed all the reading assignments up to now, including the UIMA specification, and browsed the Apache UIMA project website. You will find *Apache UIMA Manuals and Guides* (`http://uima.apache.org/documentation.html#manuals_and_guides`) will be helpful for you to finish this task. The *Tutorials and Users' Guides* gives you a step-by-step reference to build essential components for a collection processing engine.

1. After you install the UIMA Eclipse plug-in, you will need to create a collection processing engine from an archetype.
2. You might want to follow the illustrative instructions from *Tutorials and Users' Guides* to compose descriptors and Java codes.
3. Finally, you are required to describe the details of your named entity recognition algorithm, and your type system design as well as your implementation of each component. We also encourage you to report some preliminary results based on your own evaluations.

Task 3.1 Installing UIMA SDK

Similar to the installation processes you have gone through for Git and Maven, you will install the UIMA binaries, and a UIMA Eclipse plug-in. Different from the installation instruction you have learned from class or the official tutorial, you **DO NOT** need to download the UIMA SDK, uncompress and copy the archive into the Eclipse installation path. In fact, UIMA SDK will be handled by the Maven archetype `hw1-archetype` that we built for you for this task, and the Eclipse plug-in for UIMA can be installed with the Eclipse **Install New Software** function, which is basically the same as described in the official tutorial, plus one tiny additional action you have to perform to solve the dependency problem during installation.

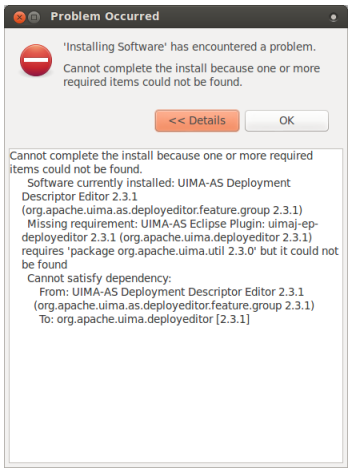

Figure 3.1: Showing a "Cannot satisfy dependency" error

1. As mentioned in the class and the official UIMA tutorial, you should install Eclipse EMF plug-in, but if strictly followed the instruction from last homework, then EMF should have been installed as it is a default component of Eclipse Juno for Java Developers. Otherwise, you should install it by yourself.

2. Follow the instruction in subsection 3.1.2[1] of the *Overview & Setup* section of *UIMA Manuals and Guides* by adding the UIMA Eclipse Plug-in Update site (`http://www.apache.org/dist/uima/eclipse-update-site`)

3. But, you might encounter a "Cannot satisfy dependency" issue like shown in Figure 3.1. You can apply a workaround for this year-long bug[2] by only selecting the "Apache UIMA Eclipse tooling and runtime support" and unselecting the "Apache UIMA-AS (Asynchronous Scaleout) Eclipse tooling" (as shown in Figure 3.2), where the latter tool will not be used for this course.

4. After installation, you are able to create or edit UIMA descriptors with a nice GUI.

Task 3.2 Creating Maven project from the archetype

For this task, we have created an archetype for you to base on, which means you don't need to manually input shared information, e.g., configurations for the Maven repositories, compile plug-in, etc. The tutorial for Homework 0 might also be helpful for you when you are creating a Maven project.

[1] `https://uima.apache.org/d/uimaj-2.4.0/overview_and_setup.html#ugr.ovv.eclipse_setup.install_uima_eclipse_plugins`

[2] You can find people asking about this issue since over a year ago at `http://www.mail-archive.com/user@uima.apache.org/msg00781.html`

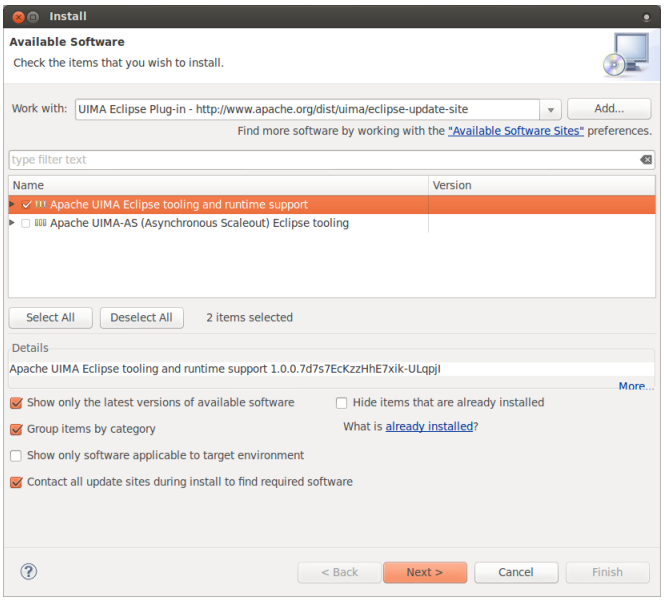

Figure 3.2: Using a workaround to solve the dependency error

1. The first thing you should do is to inform m2e where to find the archetypes. Open your Eclipse's **Preferences** window, and navigate to **Maven** → **Archetypes**, and click **Add Remote Catlog...**. (See Figure 3.3)

2. Type the following URL into the **Catalog File** field with ID and PASSWORD being your Andrew ID and Maven repository password.

   ```
   http://ID:PASSWORD@mu.lti.cs.cmu.edu:8081/nexus/content/
   groups/course/archetype-catalog.xml
   ```

   Due to page width limitation, the URL is split into two lines. Be aware that when you copy the above two lines into the text field, a space might replace the line break, so don't forget to space remove the space between `content/` and `groups`.

   Optionally, you can add a **Description** for this catalog, for example "Course Catalog". See Figure 3.4. Then click **OK** on the **Remote Archetype Catalog** window and another **OK** on the **Preferences** window.

3. Now you can follow almost the same steps to create a Maven project hosted on GitHub, you can refer to Homework 0 to find out how to create an empty project on GitHub, and how the project can be imported to Eclipse. Since we have created the archetype for you, remember to unselect **Create a simple project (skip archetype selction)** (see Figure 3.5). Then click **Next**.

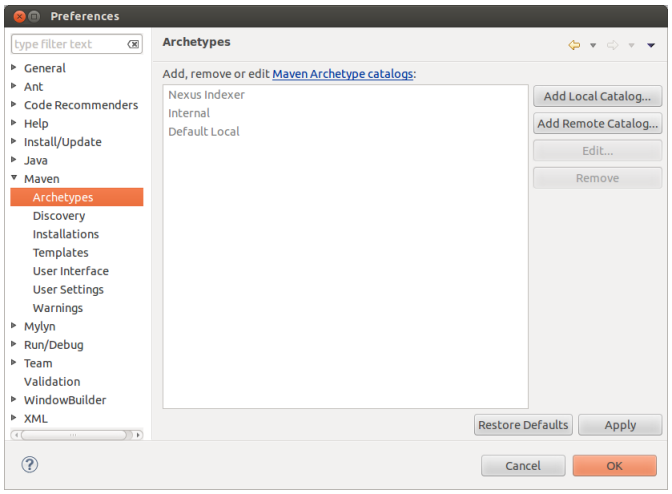

Figure 3.3: Configuring Maven catalog

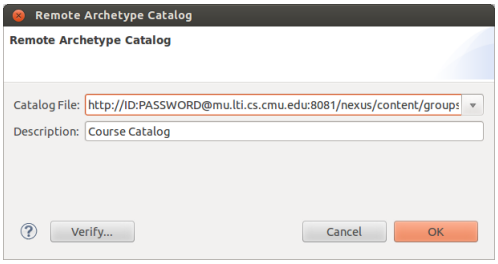

Figure 3.4: Adding a remote catalog

4. Here you can select "Course Catalog" (or other names you specified in the previous step) or "All Catalogs" in the drop-down menu for **Catalog**. Then, type in "hw1-archetype" (without quotes) in the **Filter** field. While you are typing, you will find the progress bar indicates you that it is busy "Retrieving archetypes". Select the archetype listed below, and click next to continue. See Figure 3.6.

5. In the next window, you are asked to specify the **Group Id** and **Artifact Id**. Similar to Homework 0, the Group Id is

   edu.cmu.lti.11791.f12.hw1

   and Artifact Id is

   hw1-ID

   with ID being your Andrew Id. Then click **Finish**. See Figure 3.7.

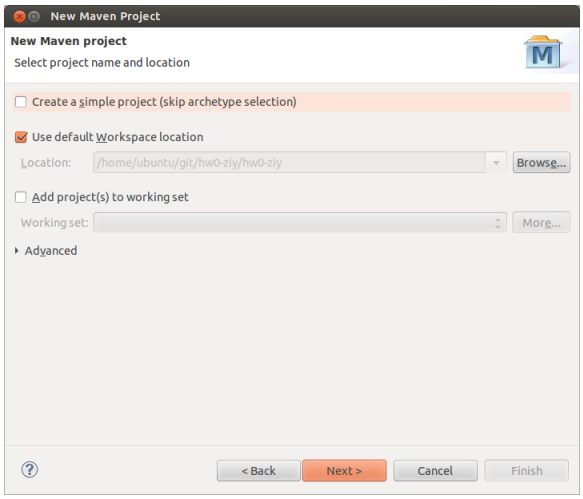

Figure 3.5: Unselecting Create a simple project

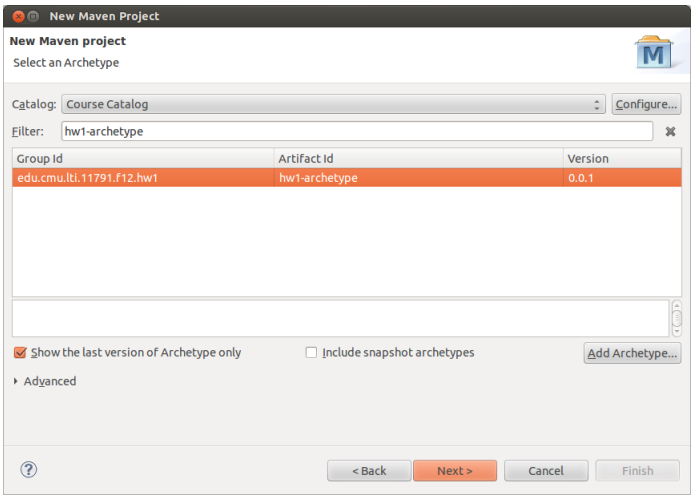

Figure 3.6: Filtering the archetype you want to base on

6. Now you have a new Maven project created from the archetype, which means there will be no extra steps to manually edit the pom.xml file on your own. Instead the archetype includes most information Maven needs to execute goals.

   The only information that Maven doesn't know about from archetype or the information we have typed to create the project is SCM. You should edit the `pom.xml` file to type in the SCM information of your GitHub repository for Homework 1 as you did in Homework 0.

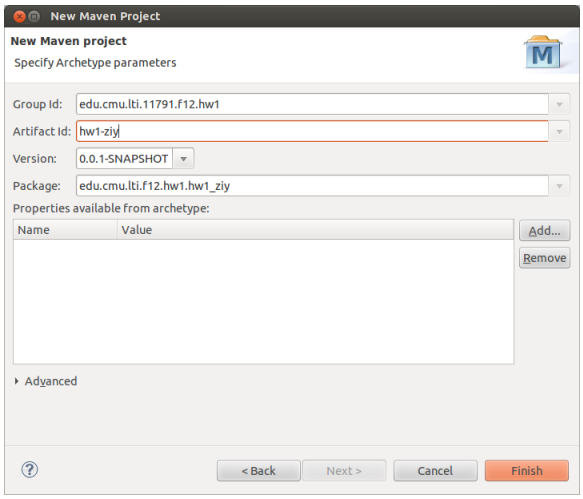

Figure 3.7: Specifying artifact parameters

You can see that we have included

- two java files in the `src/main/java` folder: `PosTagNamedEntityRecognizer.java`, which includes the algorithm that extracts name entities based on part-of-speech tags, and a `SimpleRunCPE.java`, copied from uima-examples package, which you can use as an entry point to test your CPE.
- and two files in the `src/main/resources/data` folder, `sample.in` and `sample.out`, which correspond to the sample input file and the sample output file, and you can also use them to train your model if you are trying to use a supervised approach.
- the `pom.xml`. You can go to the **Dependencies** tab after you double-click the pom file, and you will be able to see all the UIMA SDK packages have been added to your Maven project by the archetype, and they will be stored on your local Maven repository (e.g., `/.m2/repositories`). You will not need to follow the official instruction to uncompressed the SDK package and specify the `UIMA_HOME` as your environment parameter unless you want to execute a UIMA program outside your project.

Your implementation starts here.

Q1 I found a bug in the archetype and I know how to fix it. How should I proceed to patch the archetype to help many others who may suffer from this?

A1 The archetype project is also hosted on GitHub at `https://github.com/ziy/hw1-archetype`, and you can send a pull request to me regarding any issue.

Task 3.3 Adding descriptors and codes for your name entity recognizer

If you have one or several idea(s) for the Gene Mention tagging task, it's time to implement all of them in this task. You will find the official tutorial is helpful for you to accomplish this task. Except that you are required to build your component based on UIMA framework, there is little limilation on how you should design and implement your type system as well as all the pipeline components. Nevertheless, there are still some suggestions (or you can say tips or our expections) to consider.

Type system

1. The minimal type system to accomplish this task should include types for input and output elements, i.e. the id and text of each sentence and gene tags.
2. If you intend to design a pipeline of two or more phases (e.g., each of the first few phases extracts a certain feature, and the last phase predicts the gene mentions based on all the features), you should include all the intermediate types in your type system as well.
3. If your type system is complex, then you can try to refactor the type system by categorizing the types according to their functions (e.g., NLP related, biological related, etc.), and creating multiple type systems for each of the categories.
4. Another common practise in the type system design is to create a *base annotation* type which includes two features: a string feature *source* and a numeric feature *confidence* to help keep track of where an annotation was originally made from, and how confidence the annotation was. All other types then inherit from this base annotation type.
5. You could assign a (or several) namespace(s) for your types, e.g., instead of `Sentence` (default namespace), you could name it `model.Sentence` (a type with namespace `model`). Once you click the **JCasGen** button, you will find nice hierarchical folder structure created under `src/main/java` for the Java classes corresponding to the types.

Collection processing engine

1. You can use UML to help you and us clarify your pipeline design. You could include some UML diagrams in the Javadocs and in your report.

2. You can refer to the `SimpeRunCPE.java` in the uima-example package (and we also included it in the archetype) as the entry point to test your pipeline once you have done all the implementations.

3. The UIMA Eclipse plug-in has not provided a GUI for you to specify the CPE descriptor yet, which means although it will not be complicated, you still need to manually create the CPE descriptor all by yourself.

4. **Please name your CPE descriptor as CpeDescriptor.xml and put it under `src/main/resources/`**, so that we could easily find the entry point of your pipeline and locate all your component descriptors and thus your component implementations.

Collection reader

1. Since your pipeline is required to take an arbitrary file (with the same format as `sample.in`) as input, you should define a configuration parameter for the input file path in the collection reader. It could be "src/main/resources/data/sample.in" (without quotes) for your own test, but **in your final submission, please put "hw1.in" (without quotes) as the value for the parameter**, which is the path of the file used to test your pipeline.

2. Ideally, depending on where the input file is located (on a local file system, in a jar file, or from an external sources, e.g. network), your collection reader needs to be general enough to establish a connection to the file source and open a stream to read the content. But for our task, you only need to consider a file located on the file system, which means the basic `FileInputStream` and `FileReader` will do the work.

Cas consumer

1. Please refer to a prior section to find out the expected output format.

2. Similar to the collection reader, you should define a configuration parameter for the output file path. **Please name it as "hw1-ID.out" (without quotes) in your final submission, where ID is again your Andrew ID**, and then your cas consumer will write the output file to the project root directly.

3. You could create multiple views for a CAS, e.g., one for all the input elements, and one for all the intermediate results, and one for the final annotations, then your cas consumer can read the document ID from the first view, and the annotations from the last view to produce the output file. If you don't do it in this way, then you should probably include the document ID as a feature in all the intermediate types and final annotation type.

Annotator

1. If you want to employ a resource (e.g., a model you trained offline) in your annotator, you could consider UIMA's *resource manager* (refer to the official tutorial for details about this). Be sure to put your resource in `src/main/resources` so that your submission will also bundle those resources along with your codes.

2. Remember to add comments and Javadocs to your annotators, we will also evaluate the quality of your codes.

3. The most creative ideas will happen in the intermediate annotators. We have mentioned many possible solutions to do this. You can try and implement multiple approaches, and annotate the gene mentions for the sentence multiple times. All the annotations should be kept in the CAS until a final "merging" component takes all the annotations and make a final judgment.

   You can use the type's *source* feature to identify, among all the approaches, which annotator makes this annotation, and *confidence* feature as a weight to vote for a final decision. This approach was also applied in IBM's Watson system.

Task 3.4 Writing up your report

Once you test that everything works smoothly as expected, you need to summarize how your design and implement the components for the gene mention tagging task in your report.

There is not a template for the report, but we expect you could mention the system design from an archetectural aspect (UML, type system, engineering, design pattern, etc.) and algorithmic aspect (knowledge sources, NLP tools, machine learning methods, etc.), and also the preliminary experiments you conducted to evaluate the overall performance of your system (or many each of your components if you implemented multiple solutions).

If you are still not clear how to write the report. You can also consider to answer the following questions from the official Gene Mention task submission guideline:

1. Please identify/describe any machine learning techniques used:.....
2. Please identify/describe any NLP techniques/components used:.....
3. Please identify/describe any external (marked up text) training data used:.....
4. Please identify/describe any external lexical resources (terminology lists)used:.....
5. Please describe any rule sets used:.....
6. If your system interacts with or uses data from any biological database(s), please describe:.....
7. Please identify/describe any other relevant resources used to train/develop your system:.....

8. Please describe the general data flow in your system:.....

9. Other information of interest:.....

Finally, don't forget to put your name and Andrwe ID at the top of the document, name the file as "hw1-ID-report.pdf" and put it under `src/main/resources/docs`.

Appendix: GENETAG Annotation Guidelines

The following are some rules about which words are considered as part of a single gene/protein mention.

1. Mutants

   p53 mutant

2. Parens at start or end when embedded in the name

   (IGG) receptor

3. Motifs, elements, and domains **with a gene name**

   POU domain
   src domains
   RXR-responsive element
   Ras-responsive enhancer

   but not

   serum response element
   AC element
   B-cell-specific enhancer element
   dioxin responsive transcriptional enhancer

4. Plurals and families

   immunoglobulins

5. Fusion proteins

   p53 mdm2 fusion

6. The words light/heavy chain, monomer, codon, region, exon, orf, cdna, reporter gene, antibody, complex, gene product, mrna, oligomer, chemokine, subunit, peptide, message, transactivator, homolog, binding site, enhancer, element, allele, isoform, intron, promoter, operon, etc. **with a gene name**.

7. Particular qualifiers such as alpha, beta, I, II, etc. **with a gene name**

   For example, *topo* is not an allowable alternative to *topo II*

8. If the context suggests that a word is necessary, require that word in the allowable alternatives, even if it is still a name without the word.

   *rabies immunoglobulin (RIG) (**not*** immunoglobulin
   designated *HUG1*, for *Hox11 Upstream Gene* (**not** *Hox11*)

9. Viral promoters, LTRs and enhancers **with specific virus name**

   HIV long terminal repeat
   Simian virus 40 promoter
   HPV 18 enhancer

10. Antigen receptor region gene segment genes

    CH genes
    Tamarin variable region genes

11. Peptide hormones

    Vasopressin, prolactin, FSH

12. Embedded names tag **only the gene/protein part of the name**

    p53-mediated

The following generally **do not** qualify as gene name mentions:

1. Generic terms

   zinc finger alone (but *zinc finger protein* is an accepted gene/protein mention)

2. Mutations

   p53 mutations

3. Parens at start and end which 'wraps' the whole gene name

   (*IGG*)

4. TATA boxes

5. Receptors: if a receptor is specified, the gene name without "receptor" is not considered to be a valid alternative.

6. Synonyms: if a synonym is given in the sentence which implies certain words are necessary to the gene name, they will be required in the alternatives

   For *rabies immunoglobin (RIG)*, "immunoglobin" alone will not be a valid alternative because RIG implies that "rabies" is part of the name in this context.

7. Non-peptide hormones

8. DNA and protein sequences