11-791 Design and Engineering of Intelligent Information System Fall 2012
Homework 2

Configuration Space Exploration for Biomedical Question Answering

Important dates

- **Hand out: November 1.**[a]
- **Milestone 1 (M1) turn in: November 8.**
  - You are required to submit all your Java source codes and YAML descriptors for your components, and you DON'T need to submit any other documents. Your main YAML descriptor should include a single phase of multiple options for NERs.
  - In addition, please send us the URL of your project repository page (e.g., `https://github.com/ID/hw2-teamXX`). We will look into your Issues page and Wiki page, and expect you have created milestones and issues in the Issues system, and reported the results of your named entity recognizers evaluated on your own, your team meeting minutes and probably your project goals or other any important things in your Wiki page.
- **Milestone 2 (M2) turn in: November 22.**
  - You are required to submit all your Java source codes and YAML descriptors for your components, and you DON'T need to submit any other documents. You main YAML should include a complete pipeline, where each phase can only have a SINGLE component. You can pick the best component for each phase based on your own cross-option evaluation with CSE.
  - We will again look into your Issues page and Wiki page, and expect you properly solved your previous created milestones and issues, and probably created new milestones and issues, and reported the evaluation results of your M2 on your own, your team meeting minutes and probably your revised project goals.

[a]This version was built on November 5, 2012

- **Milestone 3 (M3) turn in: December 6.**
  - You are required to submit all your Java source codes and YAML descriptors for your components. You main YAML should include a complete pipeline, where each phase can only have a SINGLE component. You can pick the best component for each phase based on your own cross-option evaluation with CSE.
  - Name your presentation slides as `hw2-teamXX.ppt`[a] and put it in the `src/main/resources/docs`.
  - We expect you properly solved all the issues and accomplish all milestones, and reported the evaluation results of your final system, your team meeting minutes and a final summary.

[a] Or other formats, e.g., pptx, odp, pdf, etc.

You should always organize your project in the same hierarchy as shown below for all your submissions:

```
hw2-teamXX
|- pom.xml
|- launches
|  `- hellobioqa.launch
|- data                            /* git-ignore this folder, since
|  `- oaqa-eval.db3                 * it will become huge */
`- src
   `- main
      |- java/edu/cmu/lti/oaqa/openqa/test/teamXX
      |  `- **/*.java              /* your Java codes go into this
      |                             * folder as you did before */
      `- resources
           |- input
           |- gs
           `- hellobioqa
              |- collection        /* descriptors for collection reader,
              |- keyterm            * example keyterm extractors,
              |- retrieval          * example retrieval strategists,
              |- passage            * example passage extractors */
              |- hellobioqa.yaml   /* the entry point for the example
              |                     * pipeline and your pipeline */
              `- teamXX            /* your descriptors go into here */
                 `- **/*.yaml
```

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

1. **Submission:** The same way as you did for Homework 0 and 1 (set up GitHub repo, create Maven project, write your code, submit to Maven repo), except that the name has changed to hw2-teamXX (XX is a two-digit number from 01 to 18).

2. **Your source files and descriptors:** `java/edu/cmu/lti/oaqa/openqa/test/teamXX` means you should create directories (or packages in terms of Java program structure) iteratively from `java`, all the way to `teamXX`.

   `**/*.java` and `**/*.yaml` 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/hellobioqa/teamXX/keyterm` for all the descriptors related to keyterm extraction task.

   Actually, once you specify the main yaml descriptor of pipeline to the `ECDDriver`, you can have your own entry point to the system. However, if you want to use `launches/hellobioqa.launch` file to run the pipeline, you should consider `src/main/resources/hellobioqa/hellobioqa.yaml` as your main yaml. When we evaluate your codes, we will run a different main yaml to bundle all your components, all the components declared in your `hellobioqa.yaml` will be used.

3. **Comments, Javadocs, and documentations:** They are still important if you want us and other users to better understand your code. (Remember: we will become your customer when we run the big experiment.)

Useful information

1. Please visit the course blackboard regularly to check if a newer version is published. We may have new versions or revised instruction at the begining of each milestone. Since this is still an ongoing project, we might also fix bugs or enhance features to the framework. If an urgent bug fix is done, we will also make an announcement on the blackboard.

2. If you have difficulties in using Eclipse, Java, git, GitHub, Maven, Sonatype Nexus, Lucene, Solr, etc., you will find Google is the still the most effective way to solve any problem. But if you find the problem might exist in the framework, the quickest solution is to post an issue on the GitHub ISSUES system, and all the developers will be notified for the newly posted issue.

   Please take a look at the following table to find out where to post an issue.

Key words of your problem	Suspected project	Report it at
ECD, descriptor, phase, yaml, driver	uima-ecd	`https://github.com/oaqa/uima-ecd/issues`
CSE, persistence-provider, retrieval evaluation	cse-framework	`https://github.com/oaqa/cse-framework/issues`
QA, abstract classes, OAQA type system, keyterm evaluation, passage evaluation, collection reader	baseqa	`https://github.com/oaqa/baseqa/issues`
oaqa-eval, cse repository, database	jdbc-provider	`https://github.com/oaqa/jdbc-provider/issues`
solr, retrieval, lucene	solr-provider	`https://github.com/oaqa/solr-provider/issues`
example implementations	helloqa	`https://github.com/oaqa/helloqa/issues`
trec-genomics adaptation	hellobioqa	`https://github.com/ziy/hellobioqa/issues`

   For any question regarding the policy of Homework 2, please send mails directly to Prof. Eric Nyberg (`ehn@cs.cmu.edu`), and for other questions about the homework, 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

Getting Ready for a Biomedical Question Answering System

In this task, you won't need to write any code (except that you need to edit your Wiki page in Github flavored markdown language), instead you will learn a new biomedical question answering task. Plus, we will also mention how to create Milestones and Issues in the GitHub ISSUES system. Finally, you will create your own question answering system based on an archetype framework called `hellobioqa` we built for you.

We encourage you to go over the CSE Tutorial (which can be downloaded from the blackboard) to know the baseqa framework for the question answering framework. If you are still unsure how to create a repository on GitHub, or how to create a Maven project from an archetype, we also recommend you to take a look at Homework 0 and 1 again.

Task 1.1 Learning biomedical question answering task

Question Answering (QA) is a computer science discipline within the fields of information retrieval and natural language processing (NLP) which is concerned with building systems that automatically answer questions posed by humans in a natural language.[1] The system you are going to build will handle biomedical questions, e.g., "What is the role of PrnP in mad cow disease?". As you can see, different from a generic question answering, a biomedical QA system requires domain specific knowledge bases, NLP tools, literature collections, etc., in order to answer such questions.

One of the biomedical information retrieval benchmark is TREC Genomics Task[2]. The Text Retrieval Conference (TREC) every year provides a platform for testing and experiments of retrieval methods for different tasks. The Genomics Track for 2006 focuses on retrieval of relevant passages within an article collection from biomedical journals as answers to the questions in biomedical domain.

[1] `https://en.wikipedia.org/wiki/Question_answering`
[2] `http://ir.ohsu.edu/genomics`

Table 1.1: 28 topics (or questions) from TREC Genomics 2006

ID	Question
160	What is the role of PrnP in mad cow disease?
161	What is the role of IDE in Alzheimer's disease
162	What is the role of MMS2 in cancer?
163	What is the role of APC (adenomatous polyposis coli) in colon cancer?
164	What is the role of Nurr-77 in Parkinson's disease?
165	How do Cathepsin D (CTSD) and apolipoprotein E (ApoE) interactions contribute to Alzheimer's disease?
166	What is the role of Transforming growth factor-beta1 (TGF-beta1) in cerebral amyloid angiopathy (CAA)?
167	How does nucleoside diphosphate kinase (NM23) contribute to tumor progression?
168	How does BARD1 regulate BRCA1 activity?
169	How does APC (adenomatous polyposis coli) protein affect actin assembly
170	How does COP2 contribute to CFTR export from the endoplasmic reticulum?
171	How does Nurr-77 delete T cells before they migrate to the spleen or lymph nodes and how does this impact autoimmunity?
172	How does p53 affect apoptosis?
173	How do alpha7 nicotinic receptor subunits affect ethanol metabolism?
174	How does BRCA1 ubiquitinating activity contribute to cancer?
175	How does L2 interact with L1 to form HPV11 viral capsids?
176	How does Sec61-mediated CFTR degradation contribute to cystic fibrosis?
177	How do Bop-Pes interactions affect cell growth?
178	How do interactions between insulin-like GFs and the insulin receptor affect skin biology?
179	How do interactions between HNF4 and COUP-TF1 suppress liver function?
180	How do Ret-GDNF interactions affect liver development?
181	How do mutations in the Huntingtin gene affect Huntington's disease?
182	How do mutations in Sonic Hedgehog genes affect developmental disorders?
183	How do mutations in the NM23 gene affect tracheal development?
184	How do mutations in the Pes gene affect cell growth?
185	How do mutations in the hypocretin receptor 2 gene affect narcolepsy?
186	How do mutations in the Presenilin-1 gene affect Alzheimer's disease?
187	How do mutations in familial hemiplegic migraine type 1 (FHM1) gene affect calcium ion influx in hippocampal neurons?

The genomics track provides 28 topics (or questions)[3] created by active biological researchers. Each topic fit within one of four question-oriented topic templates: the role of a gene in a disease, the effect of a gene on a biological process, how genes interact in organ function, and how mutations in genes influence biological processes. All the questions are listed in Table 1.1:

Documents for this task come from a corpus of 162,048 full-text biomedical articles. If you want to estimate the size of the corpus, 12.3 GB (uncompressed) and 3

[3]Strictly, a topic is an information need, that can be expressed literally differently as many different questions, but here we use topic and question interchangably.

GB (standard compressed). Each document is identified by its PMID (PubMed[4] ID). Postprocessing is done to eliminate as much non-article material as we could from the original HTML formats Legal spans are defined as any text ¿ 0 bytes in length between an HTML paragraph tag, which is any tag that starts with `<p` or `</p`.

Similar to what you will be asked by the BioCreative competition organizer, you are required to follow an output format for the retrieved relevant passages. The format for TREC Genomics complies with standard TREC output format: a tabular format that consists seven fields: topic number, doc ID (PubMedID), rank, score, passage start, passage length, run tag.

The evaluation for the relevant passage retrieval is measured with a variant of mean average precision (MAP). Three different levels will be applied to the final result: passage-level, document-level and aspect-level. In addition to the passage-level evaluation, document-level captures the coverage of different relevant documents, and aspect-level evaluation captures the coverage of different MeSH terms[5].

HelloBioQA: BioQA framework and testbed implementation

We will briefly describe how the HelloBioQA framework based on CSE can help build your own QA pipeline, and evaluate your components automatically.

The input file "trecgen06.txt" has been put into the `src/main/resources/input` directory, while consists of all the questions, one per line. Each sentence will be preceded on the same line by a sentence identifier, i.e.,

```
160|What is the role of PrnP in mad cow disease?
```

The gold-standard relevant passages have also been included in HelloBioQA framework (`src/main/resources/gs/trecgen06.passage`) project, in addition, we manually annotated keyterms for the questions. Different from homework 1, the character offsets DO consider whitespaces, which is adopted by the UIMA framework. The gold-standard keyterms include not only the named entities, but also other important key verbs. Although gold-standard keyterm does not perfectly fit the requirement of NER evaluation, it will be used to test your NER components (`src/main/resources/gs/trecgen06.keyterm`). We won't consider the evaluation result in M1 when grading your homework.

We have implemented everything you need for collection reader, gold standard decorator, and evaluators, which means you don't need to put any effort to investigate how to read the questions and generate the output format in this homework. But we still encourage you to investigate how three different levels of MAP evaluation work. At the end of your pipeline execution, you will see the evaluation results similar to the following:

```
 ------------------ EVALUATION REPORT ------------------
Experiment: ac146544-b116-4cce-897e-77bfa1b4fc18:1
Evaluator,Configuration,Precision,Recall,F-1,MAP,Binary Recall,Count
```

[4] `http://www.ncbi.nlm.nih.gov/pubmed`
[5] `http://www.ncbi.nlm.nih.gov/mesh`

```
RetrievalMeasuresEvaluator,1|SimpleKeytermExtractor[persistence-provi
der:inherit: ecd.default-log-persistence-provider]>2|SimpleSolrRetrie
valStrategist[hit-list-size:10#embedded:true#core:data/guten#persiste
nce-provider:inherit: ecd.default-log-persistence-provider ]>3|Simple
PassageExtractor[hit-list-size:10#embedded:true#core:data/guten#keyte
rmWindowScorer:edu.cmu.lti.oaqa.openqa.hello.passage.KeytermWindowSco
rerSum#persistence-provider:inherit: ecd.default-log-persistence-prov
ider],0.5000,0.6667,0.5714,0.6667,1.0000,1

 ----------------------------------------------------
 ------------------ EVALUATION REPORT ------------------
Experiment: ac146544-b116-4cce-897e-77bfa1b4fc18:1
Evaluator,Configuration,Precision,Recall,F-1,MAP,Binary Recall,Count
RetrievalMeasuresEvaluator,1|SimpleKeytermExtractor[persistence-provi
der:inherit: ecd.default-log-persistence-provider]>2|SimpleSolrRetrie
valStrategist[hit-list-size:10#embedded:true#core:data/guten#persiste
nce-provider:inherit: ecd.default-log-persistence-provider ]>3|Simple
PassageExtractor[hit-list-size:10#embedded:true#core:data/guten#keyte
rmWindowScorer:edu.cmu.lti.oaqa.openqa.hello.passage.KeytermWindowSco
rerSum#persistence-provider:inherit: ecd.default-log-persistence-prov
ider],0.2000,0.3333,0.2500,0.3056,1.0000,1

 ----------------------------------------------------
 ------------------ EVALUATION REPORT ------------------
Experiment: ac146544-b116-4cce-897e-77bfa1b4fc18:1
Evaluator,Configuration,DocumentMAP,PassageMAP,AspectMAP,Count
PassageMAPMeasuresEvaluator,1|SimpleKeytermExtractor[persistence-prov
ider:inherit: ecd.default-log-persistence-provider]>2|SimpleSolrRetri
evalStrategist[hit-list-size:10#embedded:true#core:data/guten#persist
ence-provider:inherit: ecd.default-log-persistence-provider ]>3|Simpl
ePassageExtractor[hit-list-size:10#embedded:true#core:data/guten#keyt
ermWindowScorer:edu.cmu.lti.oaqa.openqa.hello.passage.KeytermWindowSc
orerSum#persistence-provider:inherit: ecd.default-log-persistence-pro
vider],0.3056,0.0202,1.0000,1
```

Three big tables correspond to the evaluation at different phases, the first table is the result for keyterm extraction, and the sencond table for relevant document retrieval, and the last table is for the final passage extraction. If you have more than traces running (e.g., multiple options for NER), then you will see multiple lines for each of the traces. If you are familiar with information retrieval, then the evaluation for keyterm extraction and document retrieval follows the standard definitions for Precision/Recall/F-1/MAP/etc., if you are unfamiliar, we suggest you to search them in Wikipedia or other information retrieval textbooks. Believe me, all of them are easy to understand! The evaluation for passage extraction follows a variant of MAP definition, you can find detailed algorithm from the task's homepage.

What you need to implement is only the `pipeline` (if you are wondering why I am using typewriter font for "pipeline". please take a look at the CSE Tutorial again). The pipeline will consist of three major phrases, which have been defined in BaseQA framework. (If you couldn't remember what they are, I again encourage you

to go over the CSE Tutorial.) In fact, you can define totally different phrases and put them into your QA pipeline[6], but for this task, you are recommended to focus on this "traditional" pipeline design, so that we can easily bundle your components for the big experiment.

Before you think about the architecture and engineering, it's a good time to have a discussion among the team members and talk about what you can do to improve the performance based on the baseline implementation given by the HelloQA project. Put them in your Wiki page, and in the next subtask, you will learn how to create the Wiki.

Q1 I've looked into the baseline implementations in HelloQA. Can you give us some more ideas on how to improve the system?

A1 Actually, since it is a past TREC task, you can easily find the papers from participants that describe how they tackled it. Another paper you might be interested in is the task overview paper from the organizer. They summarized many algorithms, and knowledge sources, when the competition was over: `http://trec.nist.gov/pubs/trec15/papers/GEO06.OVERVIEW.pdf`.

Q2 Where is the type system for HelloBioQA?

A2 As type system is a task specific, you should look for the one called `OAQATypes.xml` in `BaseQA`.

Task 1.2 Team coding with GitHub

You will again use GitHub to host your project code. But different from previous homeworks, all the team members need to collaborate on the project, and commit the code changes to the same repository, which means it is good to learn how to more about git and GitHub in this subtask.

Creating a repository

1. The team leader needs to create a repository with his/her GitHub acccount, and name your project as `hw2-teamXX`, where `XX` is your team number, which is a two-digit number ranging from 01 to 18. If you are not sure how to create a GitHub repository, please refer to Homework 0.

   At this point, all the team members are able to checkout the empty project and start working on their own. But this is NOT recommended! We recommend the team leader could checkout the project from GitHub repository, go through the entire Maven project building process (we will come to this in the next subtask) until you can ran a simple hellobioqa pipeline. Then, the team leader again,

[6]For example, Statistical Source Expansion for Question Answering, `http://dl.acm.org/citation.cfm?id=2063632`

from his/her laptop, commits and pushes everything to the repository before the team members clone the project.

2. Now, everyone has the read permission to your project repository (since it is a public repository). To help your team members gain read/write permissions, you, the team leader, should add them as collaborators. You need to click the **Admin** tab on your project homepage, and click the **Collaborators** menu on the left. Put in your teammates' GitHub IDs one by one, and click **Add** button.

Creating milestones, issues, and Wiki pages

To create milestones for your development will help better organize your team members, know your collaborators' progress, and help users/customers know what they could expect from a future release. Issues can be detailed action items team members would like to contribute to each milestone. Action can be bug fix, feature enhancement, etc. If you are still unclear what a milestone is or what an issue is, or you want to understand how milestone and issue tracking system can help software development, you can search on Google and find out many interesting blogs. You are recommended to read about the features of GitHub issue tracking system at `https://github.com/blog/831-issues-2-0-the-next-generation` and `https://github.com/features/projects/issues`.

We recommend all the team members have a discussion on how to set the milestones, and what your first several initial issues will be (i.e., what they first couple of things you want to do at the beginning.) And the team leader does the following steps. **Remember that you are unlikely to submit all the issues to the tracking system at once, you may create new issues for bugs to fix or new features to implement, change the owner (assignee) of the issue, close implemented issues, or mark some as wontfix. All your team members are responsible to maintain your milestones and issues.**

1. To create a milestone, you can navigate to the **Issues** tab of your project homepage, then click the **Milestones** below the main menu bar. Now you are able to see the button **Create a new milestone**, click it.

2. Type your title for the milestone, e.g., "M1" for the first milestone (you can also make it more meaningful and specific). Write a few descriptions for this milestone, like the goal and the brief descriptions of features will be enhanced in this milestone. Finally, select a "Due Data" for the milestone. Click the "Create Milestone" to finalize creating your milestone.

3. Do the previous step once again until all your milestones are created.

4. Now, you need to submit your first several issues. Go back to the **Issues** tab on your project homepage, and click **New Issue**. Type a title, assign the task to a particular team member, link this issue to a previously created milestone. Finally write down detailed comments to the issue.

You may find when you type "@" followed by your collaborator's ID in the comment box, you are able to mention your collaborator as you mention your friend in a tweet on Twitter.

You can learn how to write in GitHub Flavored Markdown language, by clicking the link above the textbox.

You can also attch labels to each issue by selecting them from the right panel. As we mentioned earlier, you can change the milestone assignment, in particular, if you find you couldn't finish it by M1, then you can change it to M2 later, or you can also relabel it as **wontfix**.

Now, you can create a Wiki page for your team meeting minutes and other important items.

1. Click the "Wiki" tab at the top of your project homepage, and click **Edit Page** to start editing it.

Once you reach this point, remember to send us an email to report the URL of your project repository page (e.g., `https://github.com/ID/hw2-teamXX`).

Team coding with git-branch, git-merge, git-rebase

Collaboration is important for this homework. All the team members start their individual development after the team leader gets the framework ready, and pushes all his/her local commits to the repository. Once a team member finalizes his/her development, he/she might take responsibility on another task, or want to check the integrity or compatibility with features implemented by collaborators. After all the individual developments are done, team leader is responsible to merge all the newly developed components into the same codebase and test the integrity.

Git branching is a good tool to help the team manage parallel development and distributed codebase. In fact, the branching mechanism is widely adopted by not only git, but many other version control systems, e.g., SVN or Mercurial (Hg). But one of the most important reasons that people love git branching over SVN or Mercurial (Hg) is its light-weight nature, which allow switching between development branches within the same clone of a repository.

Normally, the team leader is in charge of the `master` branch (the one you probably used for homework 0 and 1), which usually corresponds to a codebase for the most recent stable release, while team members should create branches for each individual task assignment, e.g., bug fixes, feature enhancement with `git branch` command. You can also do it within Eclipse by right-clicking the project name, and select **Team** $\rightarrow$ **Switch To** $\rightarrow$ **New Branch...**.

To merge multiple development branches back to `master` branch (or in Eclipse **Team** $\rightarrow$ **Switch To** $\rightarrow$ **master**), you should switch back to master, and then `git merge` the branch you want to be merged (or **Team** $\rightarrow$ **Merge...**). Sometimes, you may also need `git rebase` when you later realize your development should depend on another feature that was also being developed by your collaborator, which hadn't

```
<server>
  <id>oaqa</id>
  <username>ID</username>
  <password>PASSWORD</password>
</server>
```

Listing 1.1: Configuring settings.xml

beed integrated in the `master` branch by the time you created your development branch.

To better understand git branching, you should read the "Git Branching" chapter of Pro Git Book (`http://git-scm.com/book/en/Git-Branching`). You can also find a more sophisticated branching model at `http://nvie.com/posts/a-successful-git-branching-model/`, which may be too complicated for this homework, but it can inspire how you want to manage your branches.

If you are looking for a Eclipse plug-in that can bring the GitHub issue tracking system to your workspace, e.g., create/close/comment issues, label them, assign to a person within eclipse, or automatically get notified by a message bubble if an issue is assigned to you, then you can try to investigate Mylyn plug-in. In the "Tips and Tricks using Eclipse with Github" post[7] , you will find the basic idea how Mylyn GitHub connector works. By default, Eclipse Juno for Java developers comes with EGit, Mylyn, and Mylyn Github connector, and the Task View is on the top-right corner of the Java perspective by default.

Task 1.3 Creating Maven project from the archetype

For this homework, we create another archetype called `hellobioqa-archetype` to help you quickly get your development started. We briefly show you the process you've gone through for your Homework 1.

1. Open your Eclipse's **Preferences** window, and navigate to **Maven** → **Archetypes**, and click **Add Remote Catalog...**.

2. Type the following URL into the **Catalog File** field.

   `http://ziy.github.com/hellobioqa-archetype/repository/archetype-catalog.xml`

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

3. Include the following lines in your `settings.xml` in order to download the artifact if you didn't do so in Homework 1.

[7] `http://eclipsesource.com/blogs/2012/08/28/tips-and-tricks-using-eclipse-with-github/`

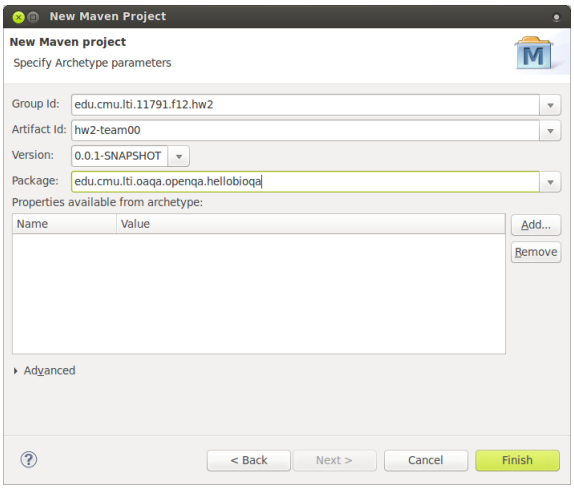

Figure 1.1: Parameters to create a Maven artifact from archetype

4. Now you can follow almost the same steps to import to Eclipse as you did for Homework 1. Since we have created the archetype for you, remember to unselect **Create a simple project (skip archetype selction)**. Then click **Next**.

5. Here you can select "HelloBioQA Catalog" (or other names you specified in the previous step) or "All Catalogs" in the drop-down menu for **Catalog**. Then, type in "hellobioqa-archetype" (without quotes) in the **Filter** field, and in order to get the latest snapshot archetypes, you need to check **Include snapshot archetypes** as well. Select the archetype listed below, and click next to continue.

6. In the next window, you are asked to specify the **Group Id** and **Artifact Id**. Similar to Homework 0 and 1, the Group Id is

   edu.cmu.lti.11791.f12.hw2

   and Artifact Id is

   hw2-teamXX

   with XX being your team number. Since we also included two sample components for retrieval strategist and passage extraction within a particular, remember to specify `Package` as

   edu.cmu.lti.oaqa.openqa.hellobioqa

   See Figure 1.1. Then click **Finish**.

7. You need to edit the `pom.xml` file to type in the SCM information of your GitHub repository for Homework 2 as you did in Homework 0.

8. You also need to manually edit the `launches/hellobioqa.launch` file. Open the file, replace the `${project_name}` with your project name (e.g., `hw2-team00`).

9. Probably you want to add another line into launch file to give extra memory to run the pipeline, like the following

   ```
   <stringAttribute key="org.eclipse.jdt.launching.VM_ARGUMENTS" value="-Xmx1g'
   ```

10. Same as before, you probably need to right-click the project name, and click **Maven** → **Update Project** to download the dependencies.

You can see that we have included

- the `pom.xml`. You can go to the **Dependencies** tab after you double-click the pom file, and you will be able to see your project only depends on `helloqa` project, which brings simple implementations for all the phases described in `baseqa` project. You won't need any direct dependency on any UIMA SDK package, which are essential to your Homework 1.

  If you want to take a look at what are inside `helloqa` project and other indirect dependencies, you can unfold the `Maven Dependencies` folder under your project name in the Package Explorer View.

- input file in the `src/main/resources/input` folder, and gold-standard annotations in the `src/main/resources/gs` folder. If you are interested in their contents, you can open them with a text editor.

- several descriptors under `src/main/resources/hellobioqa` directory and subdirectories. All of them are specific to the hellobioqa task, and actually, you can find more descriptors from dependencies, e.g., `helloqa`, `jdbc-provider`, etc. These descriptors can also be considered to incorporate in your project.

- the main yaml `src/main/resources/hellobioqa/hellobioqa.yaml`

- an Eclipse launch file `launches/hellobioqa.launch`. To run the pipeline, you can right-click this file in the Package Explorer View, and then click **Run As** → **hellobioqa**.

- a local database version of the cse repository `data/oaqa-eval.db3`. All your experiment intermediate result and evaluation results will be permanently stored here (unless you manually delete table entries), and you will find the file may grow to several megabytes after thundreds of experiments. Therefore, we suggest you should not commit this file to the GitHub repository, instead you can `git ignore` this file to avoid this huge file bothering you every time you commit your project.

Before you commit and push all the initial code changes to GitHub repository, we suggest you to first test if you can successfully run the pipeline.

1. Right-click this file in the Package Explorer View, and then click **Run As** → **hellobioqa**. Wait until all 28 questions are processed, and the evaluation results are printed to the console, and you find no exceptions are thrown.

2. Git-ignore `data/oaqa-eval.db3`[8], and commit/push all other code changes to the GitHub repository. Remember to commit the .project file, .classpath file and all other important configuration files from your project root directory, and it will be helpful for your team members to directly import an existing project.

3. Now, other team members are able to clone the repository to their workspaces and start working on particular task. When you are asked to **Select a wizard to use for importing projects**, don't forget to select **Import existing projects** as long as your team leader commited .project and .classth to the repository.

[8]Since your team member also needs the database file to run experiment, the team leader can add it to the index first then remove it after team members clone the project to their machines, or all the members can download an empty file from here: `https://github.com/oaqa/helloqa/blob/a051e1233be92bca309ff8761835fce412f6bfd5/data/oaqa-eval.db3?raw=true`, and put it under `data/`.

Task 2

Migrating from UIMA SDK to CSE Framework

In this task (for Milestone 1), you won't need to write too much code (some changes to your existing annotators may be necessary), instead you will be guided to migrate from UIMA SDK to CSE Framework by putting all the named entity recognizers you team members implemented in Homework 1 into the new `hellobioqa` project and listing all of them as options.

Task 2.1 Planning on keyterm extractor based on abstract classes

Different from what you did for Homework 1, where you built the whole UIMA CPE pipeline all by yourself, by composing `Annotator`s and writing XML descriptors, UIMA ECD & CSE framework provide you with easy-to-write YAML based descriptors and easy-to-use execution module. Moreover, `BaseQA` project includes abstract classes that extend from `JCasAnnotator_ImplBase`. For this task, you need to focus on two abstract classes defined for the first phrase *keyterm extraction*: `AbstractKeytermExtractor` and `AbstractKeytermUpdater`.

Based on your experience with UIMA SDK, there are two subtasks to migrate your code from UIMA SDK to CSE: Java code migration and descriptor migration. We will help you with both of them in the next sections. But before you move onto the next section, we encourage the teams should clearly plan and identify the tasks (by creating issues on GitHub and assigning each to a particular team member) before start coding. For example, if you have four members in the team, then it might be a good idea to create four different issues indicating four different integration subtasks.

Q1 What if all of us implemented exactly the same algorithm for Homework 1? Do we still need to incorporate all of them?

A1 Yes. For M1, it is just a practise for all the team members, and you should report the comparison experiment even though some of them might show the

same performance in evaluation results. If you want to see a different result, we encourage you to include the baseline NER method we provided to you in hw1-archetype, and integrate it as another option.

For M2 and M3, you only need to incorporate a single best component for each phrase.

Q2 I notice that the gold-standard keyterms are not only gene or protein names, and it also makes sense to me that keyterms should contain other types of important information, like key verbs. But our implementation for Homework 1 focused on gene and protein name identification, should I make any change to my code to in order to identify other types of keyterms?

A2 Generally speaking, you don't need for M1. We won't grade your M1 based on keyterm extraction results. But for M2 and M3, you may find sometimes gene and protein names are not enough to express the information need.

Task 2.2 Inheriting from AbstractKeytermExtractor or AbstractKeytermUpdater

To migrate each of your team members' components, you can do the following:

1. As suggested earlier, you should create your own components in the package `java/edu/cmu/lti/oaqa/openqa/test/teamXX/keyterm`, in case another `teamYY` might have a component with exactly the same class name as yours.

2. `AbstractKeytermExtractor` is mandatory for the pipeline, which identifies the keyterms from a question. Most of the time, you might want to inherit from `AbstractKeytermExtractor` for your own `KeytermExtractor` (please think of a better name for your component rather than `KeytermExtractor`, `MyKeytermExtractor`, or `TeamXXNKeytermExtractor`).

   In Listing 2.1, in you can see the only method your component needs to implement (the hook method) is `getKeyterms`, and the `process` method helps you retrieve data you need from CAS and store your output back to CAS.

3. Since your component extends `JCasAnnotator_ImplBase`, and `process` is the only method finalized by the abstract class, which means if you can still `getConfigParameterValue` within an `@Override`-ing `initialize` method. But remember to call the parent's `initialize` method before look up your own configuration parameters. Specifically, you can write the `initialize` method similar to this:

4. `Keyterm` is defined in the **BaseQA** project, and it is easy to create by using the constructor `Keyterm(String text)`.

```
1  public abstract class AbstractKeytermExtractor extends
       AbstractLoggedComponent {
3
     protected abstract List<Keyterm> getKeyterms(String question);
5
     @Override
     public final void process(JCas jcas) throws
         AnalysisEngineProcessException {
7      super.process(jcas);
       try {
9        // prepare input
         InputElement input =
11           (InputElement) BaseJCasHelper.getAnnotation(jcas,
                 InputElement.type);
         String question = input.getQuestion();
13       // do task
         List<Keyterm> keyterms = getKeyterms(question);
15       log(keyterms.toString());
         // save output
17       KeytermList.storeKeyterms(jcas, keyterms);
       } catch (Exception e) {
19       throw new AnalysisEngineProcessException(e);
       }
21   }
23   protected final void log(String message) {
       super.log(QALogEntry.KEYTERM, message);
25   }
27 }
```

Listing 2.1: AbstractKeytermExtractor.java

```
1  @Override
   public void initialize(UimaContext c)
3          throws ResourceInitializationException {
     super.initialize(c);
5    String p = (String) c.getConfigParameterValue("param1");
   }
```

Listing 2.2: Sample initialze method

```
public abstract class AbstractKeytermUpdater extends
    AbstractLoggedComponent {

  protected abstract List<Keyterm> updateKeyterms(String question,
      List<Keyterm> keyterms);

  @Override
  public final void process(JCas jcas) throws
      AnalysisEngineProcessException {
    super.process(jcas);
    try {
      // prepare input
      String question =
          ((InputElement) BaseJCasHelper.getAnnotation(jcas,
              InputElement.type))
              .getQuestion();
      List<Keyterm> keyterms;
      keyterms = KeytermList.retrieveKeyterms(jcas);
      // do task
      keyterms = updateKeyterms(question, keyterms);
      log(keyterms.toString());
      // save output
      KeytermList.storeKeyterms(jcas, keyterms);
    } catch (Exception e) {
      throw new AnalysisEngineProcessException(e);
    }
  }

  protected final void log(String message) {
    super.log(QALogEntry.KEYTERM, message);
  }

}
```

Listing 2.3: AbstractKeytermUpdater.java

5. You can also consider to use the simple keyterm extractor from `HelloQA`, and make necessary change to the extracted keyterms with one or several components that extend `AbstractKeytermUpdater`. In Listing 2.3, you can see the only abstract method is `updateKeyterms`, where you can get all the extract keyterms from a previous step as inputs, and you have to return another list of keyterms.

Task 2.3 Writing YAML descriptor

To write a descriptor for an annotator in CSE is much easier than you did in Homework 1 since it is based on YAML instead of XML. If you want to find a tool that can assist

```
class: java.edu.cmu.lti.oaqa.openqa.test.teamXX.FooKeytermExtractor

persistence-provider: |
  inherit: ecd.default-log-persistence-provider
```

Listing 2.4: Descriptor for a component without parameters

```
class: java.edu.cmu.lti.oaqa.openqa.test.teamXX.FooKeytermExtractor
foo: bar

persistence-provider: |
  inherit: ecd.default-log-persistence-provider
```

Listing 2.5: Descriptor for a component with parameters

```
class: java.edu.cmu.lti.oaqa.openqa.test.teamXX.FooKeytermExtractor
foo: bar
cross-opts:
  parameter-a: [value1, value2]
  parameter-b: [value3, value4]

persistence-provider: |
  inherit: ecd.default-log-persistence-provider
```

Listing 2.6: Descriptor for a component with cross-opts

you to edit YAML files, e.g., syntax hightlight, automatic YAML grammar checking, you can consider Eclipse YEdit plug-in.

1. Under `src/main/resources/hellobioqa` directory, create a subdirectory `teamXX` for your components, and then probably you want to create different subdirectories (`keyterm`, `retrieval`, and `passage`) in `teamXX` directory for different phases.

2. Then create the descriptor file (with extension name `yaml`) for your component. Open the descriptor file. If you installed YEdit plug-in, then you are able to right-click the file, and click **Open With** → **YEdit YAML Editor**.

3. If you component doesn't required any additional configuration parameters, then just simply type your class path and a `persistence-provider` into your file, like in Listing 2.4. Remember that the `persistence-provider` is required by all the components. For components with parameters, you can declare the parameters and values like in List 2.5. If you want to set different values to the same parameter, and test their performances in a single experiment, you can use `cross-opts` like in Listing 2.6.

```
pipeline:
  - inherit: jdbc.sqlite.cse.phase
    name: keyterm-extractor
    options: |
      - inherit: helloqa.keyterm.simple
      - inherit: hellobioqa.teamXX.keyterm.foo
      - inherit: hellobioqa.teamXX.keyterm.bar

  - inherit: jdbc.sqlite.cse.phase
    name: retrieval-strategist
    options: |
      - inherit: hellobioqa.retrieval.simple-solr-strategist

  - inherit: jdbc.sqlite.cse.phase
    name: passage-extractor
    options: |
      - inherit: hellobioqa.passage.simple

  - inherit: helloqa.eval.keyterm-aggregator-consumer

  - inherit: jdbc.eval.retrieval-aggregator-consumer

  - inherit: helloqa.eval.passage-map-aggregator-consumer
```

Listing 2.7: Pipeline descriptor

4. Finally, you need to properly set the pointers to all of them in the main yaml. Open the `src/main/resources/hellobioqa/hellobioqa.yaml`, and change the `author` to "teamXX". Add your components into the `options` fields of the first phase, to make the `pipeline` field look like Listing 2.7.

5. Run the pipeline again by right-clicking the launch file as we mentioned earlier. Then git-commit, git-push, git-checkout (change to `master` branch. Not sure? Take a look at the Pro Git Book), git-merge, maven release:prepare, maven release:perform.