# Assignment 1
11-791: DEIIS

**Submitted by: Ankur Gandhe**

September 11, 2013

# 1 Design of Logical Data model

The task of this assignment is to design a type system for a task which reads a file containing a question and a set of sentences, determines which sentences answer the question. Each line in the file is either the question or a candidate answer:

```
Questions are of the form: Q <question>.
Answers are of the form: A <isCorrect=0/1> <answer>
```

## 1.1 Design overview

We explain our design methodology according to the processing pipeline the data follows, specifying types and features that we create to handle each situation.

1. **BaseAnnotation**: To store the source of the part that created a particular annotation and its confidence, we define a base annotation , with the supertype *uima.tcas.Annotation*.

   ```
   Type: BaseAnnotation, Features: source, confidence
   ```

2. **Test Element Annotation:** The system reads an input file containing question and set of answers. Hence, we have a top level *TestElement:* type to read the file and store the questions and set of answers. It also stores the feature for evaluation type, which is explained later. Since questions and answers are going to be further operated upon, we also create an *Answer* type and a *Question* type. Both these classes have the supertype *BaseAnnotation*. The *Answer* type also has a boolean valued feature 'correctAnswer' that stores whether the answer is correct or wrong.

   ```
   Type: TestElement, Features: AnswerList, Question, evaluator
   Type: Answer, Features: correctAnswer, tokenList, score, unigramList,
   bigramList, trigramList
   Type: Question, Features: tokenList, unigramList, bigramList, trigramList
   ```

3. **Token Annotation:** In order to store the annotations of tokens in a question or answer, we define a *Token* type, which stores the text of the actual token, apart from start and end inherited from *BaseAnnotation* supertype. Then, *Answer* and *Question* types have a 'tokenList' as feature which store the tokens occurring in the two types ( as written above).

   `Type: Token, Features: text`

4. **Ngram Annotation:** The system needs to keep track of ngram-annotations of the question and the set of answers. Hence, we create three separate classes *Unigram* (for 1-gram), *Bigram* (for 2-grams) and *Trigram* (for 3-grams). Each of them have 'tokenList' as a feature to store the tokens present in the n-gram and inherit the *BaseAnnotation* class.

   `Type: Unigram, Bigram, Trigram, Features: tokenList`

5. **Answer Scoring:** Each answer needs to be assigned a score. Hence, we create a *AnswerScore* class that records the score assigned to the particular answer. As a result, *Answer* type needs to have 'score' (of type *AnswerScore*) as a feature. Also, *AnswerScore* inherits *BaseAnnotation*, so we can easily trace the source and confidence of the score annotator during debugging.

   `Type: AnswerScore, Features: score`

6. **Evaluation:** The system needs to sort the answers according to score and calculate precisionN. We define a class *Evaluator:* to score the metric of evaluation and perform the scoring. It currently has feature 'precisionAtN' and 'n', but it can be extended to other metrics as well. Since the evaluation is for the entire document, *TestElement* type has the feature 'evaluator' to evaluate and report the results.

   `Type: Evaluator, Features: precisionAtN, n`

The dependencies are represented in a concise manner in the UML diagram below.

## 1.2 Other concerns

While designing our type system, we chose one of several options available for design on several occasions.

- While designing the n-gram annotation, we had an option of creating a unified type *Ngram* with features 'n' and 'tokenList'. This would have allowed us to add larger n-grams (4,5) at a later stage, if required, without changing the type system. However, debugging and looking at specif 2-gram or 3-gram would have been harder. Hence, we made a decision to create separate n-gram annotation types for easier analysis and debugging.

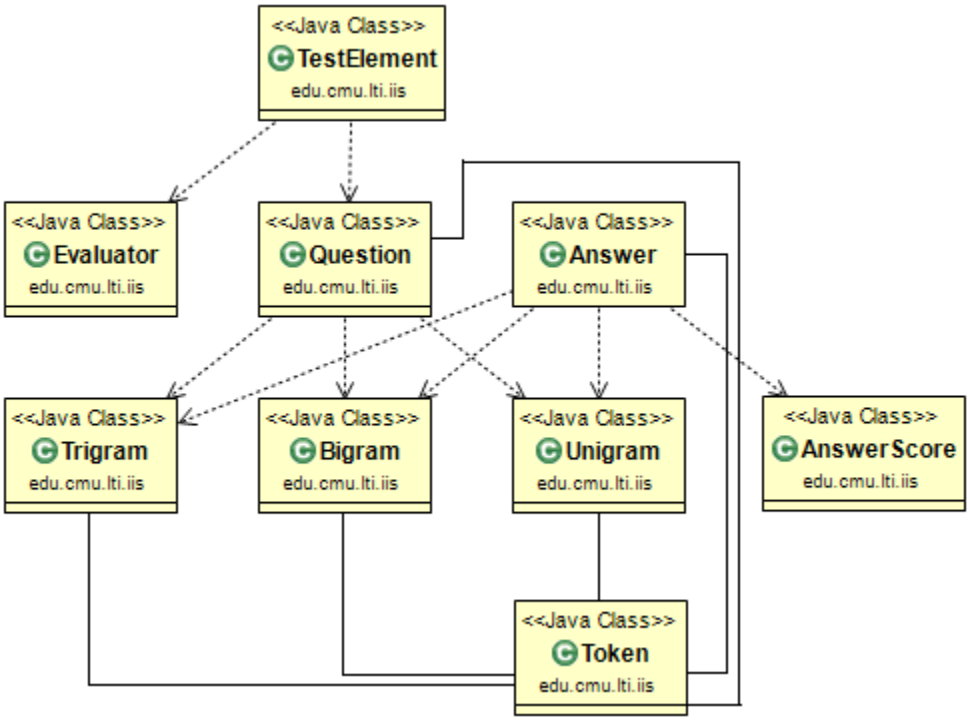


Figure 1: Basic UML diagram

- While designing the evaluation annotation, we chose to create a separate type for annotation, instead of directly adding 'precisionAtN' as a feature to the test element. This was done for easy extension to other kinds of metrics, which we may want to look at.