Design choices:

- *Static nested class WordsKey*:
  - an String[] instance variable indicating the words appearing in a sentence
  - arr[0] appears first and arr[size-1] appears last
- *Instance variables:*
  - private int n;
    - n-gram
  - private int KSmoothing;
    - add-K smoothing
  - private double[] linearInterpoParams = null;
    - parameters of Linear Interpolation
  - private int lowFrequencyThreshold;
    - low frequency threshold for Named Entity Recogtion
  - private int unkThreshold;
    - frequency threshold for converting words into UNK
  - private Map<WordsKey, Long>[] countContainer;
    - the count of a certain WordsKey in the training set
    - length of this array is n-gram
  - private Map<WordsKey, Double>[] modelParams;
    - parameters of a n-gram model
    - arr[ngram-1] is the one used for making predictions
    - length of this array is n-gram
  - private TreeSet<String> vocabulary;
    - final vocabulary of this model
  - private Map<String, String> mappingList;
    - a mapping from low-frequency named entity to categories
- *train() method:*
  - traverse the whole training set once and update counts for countContainer[]
    - Include <START> and <STOP>
  - do the named entity recognition
    - update conuntContainer[0] based on the lowFrequencyThreshold
    - create a mappingList
  - convert words into UNK based on the unkThreshold
    - Update countContainer[0] and create the final vocabulary
    - update the remaining countContainer[j] based on mappingList and vocabulary
  - Estimate the parameters using Maximum Likelihood Estimate
  - If there is Linear Interpolation, update modelParams[ngram-1]
- *predict() method:*
  - create a WordsKey when traversing the document and get the parameter
    - add <START> if necessary for the lookup in the countContainer[]
  - If there doesn't exist that parameter, make an estimate based on MLE
- *evaluate() method:*
  - output the perplexity given a list of log predictions