



# OPENNLP

**tutorialspoint**

SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

---

Apache **OpenNLP** is an open source Java library which is used process Natural Language text. OpenNLP provides services such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co-reference resolution, etc.

In this tutorial, we will understand how to use the OpenNLP library to build an efficient text processing service.

## Audience

---

This tutorial has been prepared for beginners to make them understand how to use the OpenNLP library, and thus help them in building text processing services using this library.

## Prerequisites

---

For this tutorial, it is assumed that the readers have a prior knowledge of Java programming language.

## Copyright & Disclaimer

---

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com)

## Table of Contents

---

About the Tutorial.....	i
Audience.....	i
Prerequisites.....	i
Copyright & Disclaimer.....	i
Table of Contents .....	ii
<b>1. OPENNLP – OVERVIEW .....</b>	<b>1</b>
What is Open NLP? .....	1
Features of OpenNLP.....	1
Open NLP Models.....	3
<b>2. OPENNLP – ENVIRONMENT .....</b>	<b>5</b>
Installing OpenNLP .....	5
Setting the Classpath.....	7
Eclipse Installation .....	9
<b>3. OPENNLP – REFERENCED API.....</b>	<b>16</b>
Sentence Detection .....	16
Tokenization.....	17
NameEntityRecognition .....	18
Finding the Parts of Speech .....	18
Parsing the Sentence .....	19
Chunking .....	20
<b>4. OPEN NLP – SENTENCE DETECTION.....</b>	<b>21</b>
Sentence Detection Using Java .....	21
Sentence Detection Using OpenNLP.....	22
Detecting the Positions of the Sentences.....	24
Sentences along with their Positions.....	27

Sentence Probability Detection .....	28
<b>5. OPEN NLP – TOKENIZATION .....</b>	<b>30</b>
Tokenizing using OpenNLP .....	30
Retrieving the Positions of the Tokens .....	36
Tokenizer Probability .....	41
<b>6. OPEN NLP – NAMED ENTITY RECOGNITION .....</b>	<b>44</b>
Named Entity Recognition using open NLP .....	44
Names along with their Positions .....	47
Finding the Names of the Location .....	48
NameFinder Probability .....	50
<b>7. OPENNLP – FINDING PARTS OF SPEECH .....</b>	<b>52</b>
Tagging the Parts of Speech .....	52
POS Tagger Performance .....	55
POS Tagger Probability .....	57
<b>8. OPENNLP – PARSING THE SENTENCES .....</b>	<b>59</b>
Parsing Raw Text using OpenNLP Library .....	59
<b>9. OPENNLP – CHUNKING SENTENCES .....</b>	<b>62</b>
Chunking a Sentence using OpenNLP .....	62
Detecting the Positions of the Tokens .....	65
Chunker Probability Detection .....	67
<b>10. OPENNLP – COMMAND LINE INTERFACE .....</b>	<b>70</b>
Tokenizing .....	70
Sentence Detection .....	71
Named Entity Recognition .....	71
Parts of Speech Tagging .....	72

# 1. OpenNLP – Overview

NLP is a set of tools used to derive meaningful and useful information from natural language sources such as web pages and text documents.

## What is Open NLP?

---

Apache **OpenNLP** is an open-source Java library which is used to process natural language text. You can build an efficient text processing service using this library.

OpenNLP provides services such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co-reference resolution, etc.

## A Brief History of OpenNLP

- In 2010, OpenNLP entered the Apache incubation.
- In 2011, Apache OpenNLP 1.5.2 Incubating was released, and in the same year, it graduated as a top-level Apache project.
- In 2015, OpenNLP 1.6.0 was released.

## Features of OpenNLP

---

Following are the notable features of OpenNLP –

- **Named Entity Recognition (NER):** Open NLP supports NER, using which you can extract names of locations, people and things even while processing queries.
- **Summarize:** Using the **summarize** feature, you can summarize Paragraphs, articles, documents or their collection in NLP.
- **Searching:** In OpenNLP, a given search string or its synonyms can be identified in given text, even though the given word is altered or misspelled.
- **Tagging (POS):** Tagging in NLP is used to divide the text into various grammatical elements for further analysis.
- **Translation:** In NLP, Translation helps in translating one language into another.
- **Information grouping:** This option in NLP groups the textual information in the content of the document, just like Parts of speech.
- **Natural Language Generation:** It is used for generating information from a database and automating the information reports such as weather analysis or medical reports.



- **Feedback Analysis:** As the name implies, various types of feedbacks from people are collected, regarding the products, by NLP to analyze how well the product is successful in winning their hearts.
- **Speech recognition:** Though it is difficult to analyze human speech, NLP has some built-in features for this requirement.

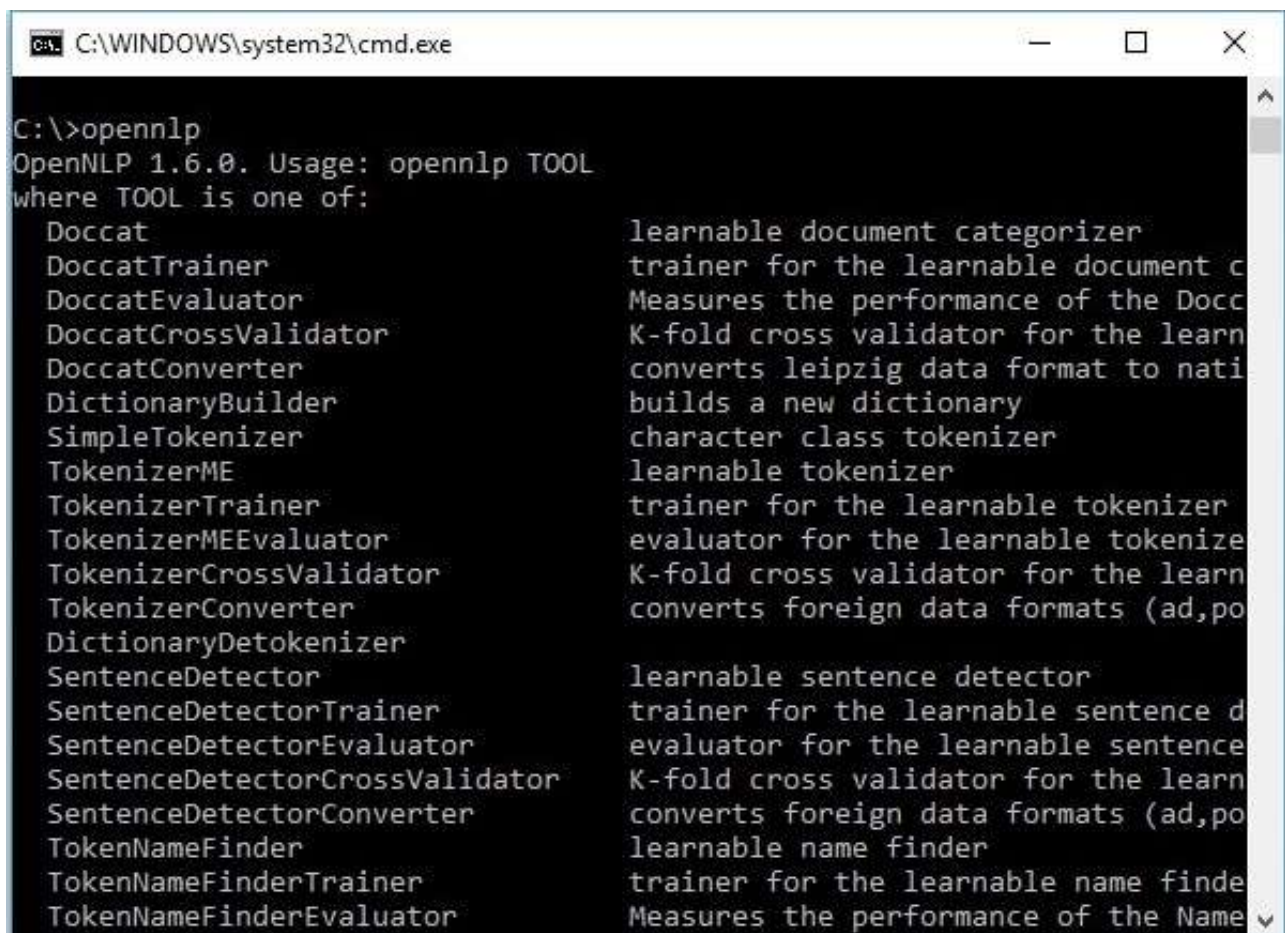
## Open NLP API

The Apache OpenNLP library provides classes and interfaces to perform various tasks of natural language processing such as sentence detection, tokenization, finding a name, tagging the parts of speech, chunking a sentence, parsing, co-reference resolution, and document categorization.

In addition to these tasks, we can also train and evaluate our own models for any of these tasks.

## OpenNLP CLI

In addition to the library, OpenNLP also provides a Command Line Interface (CLI), where we can train and evaluate models. We will discuss this topic in detail in the last chapter of this tutorial.



```

C:\WINDOWS\system32\cmd.exe

C:\>opennlp
OpenNLP 1.6.0. Usage: opennlp TOOL
where TOOL is one of:
  Doccat                learnable document categorizer
  DoccatTrainer          trainer for the learnable document c
  DoccatEvaluator        Measures the performance of the Docc
  DoccatCrossValidator   K-fold cross validator for the learn
  DoccatConverter        converts leipzig data format to nati
  DictionaryBuilder      builds a new dictionary
  SimpleTokenizer        character class tokenizer
  TokenizerME            learnable tokenizer
  TokenizerTrainer       trainer for the learnable tokenizer
  TokenizerMEEvaluator   evaluator for the learnable tokenize
  TokenizerCrossValidator K-fold cross validator for the learn
  TokenizerConverter     converts foreign data formats (ad,po
  DictionaryDetokenizer
  SentenceDetector        learnable sentence detector
  SentenceDetectorTrainer trainer for the learnable sentence d
  SentenceDetectorEvaluator evaluator for the learnable sentence
  SentenceDetectorCrossValidator K-fold cross validator for the learn
  SentenceDetectorConverter converts foreign data formats (ad,po
  TokenNameFinder         learnable name finder
  TokenNameFinderTrainer  trainer for the learnable name finde
  TokenNameFinderEvaluator Measures the performance of the Name
  
```

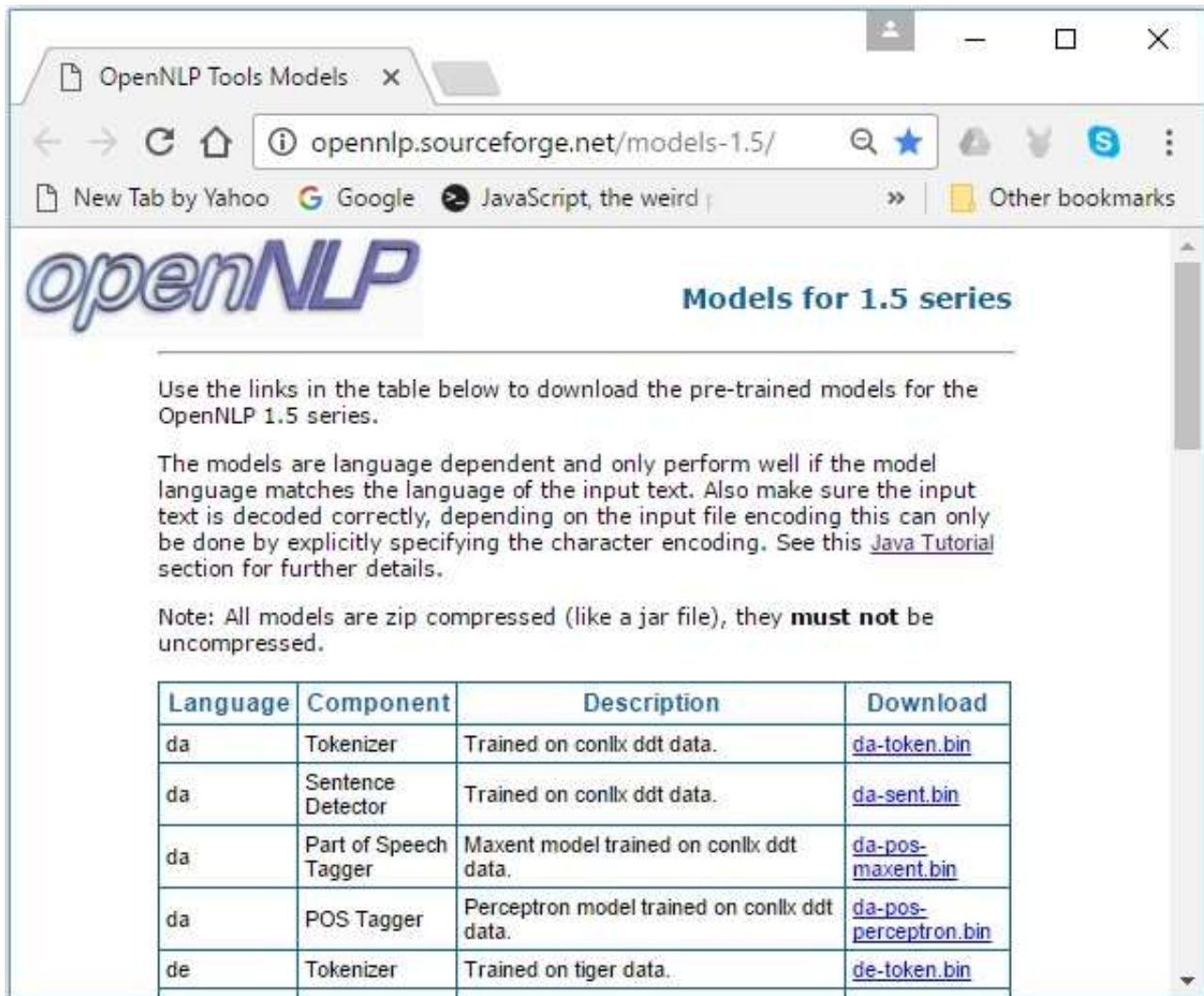
## Open NLP Models

To perform various NLP tasks, OpenNLP provides a set of predefined models. This set includes models for different languages.

### Downloading the models

You can follow the steps given below to download the predefined models provided by OpenNLP.

**Step 1:** Open the index page of OpenNLP models by clicking the following link: <http://opennlp.sourceforge.net/models-1.5/>



Use the links in the table below to download the pre-trained models for the OpenNLP 1.5 series.

The models are language dependent and only perform well if the model language matches the language of the input text. Also make sure the input text is decoded correctly, depending on the input file encoding this can only be done by explicitly specifying the character encoding. See this [Java Tutorial](#) section for further details.

Note: All models are zip compressed (like a jar file), they **must not** be uncompressed.

Language	Component	Description	Download
da	Tokenizer	Trained on conllx ddt data.	<a href="#">da-token.bin</a>
da	Sentence Detector	Trained on conllx ddt data.	<a href="#">da-sent.bin</a>
da	Part of Speech Tagger	Maxent model trained on conllx ddt data.	<a href="#">da-pos-maxent.bin</a>
da	POS Tagger	Perceptron model trained on conllx ddt data.	<a href="#">da-pos-perceptron.bin</a>
de	Tokenizer	Trained on tiger data.	<a href="#">de-token.bin</a>

**Step 2:** On visiting the given link, you will get to see a list of components of various languages and the links to download them. Here, you can get the list of all the predefined models provided by OpenNLP.

en	Tokenizer	Trained on opennlp training data.	<a href="#">en-token.bin</a>
en	Sentence Detector	Trained on opennlp training data.	<a href="#">en-sent.bin</a>
en	POS Tagger	Maxent model with tag dictionary.	<a href="#">en-pos-maxent.bin</a>
en	POS Tagger	Perceptron model with tag dictionary.	<a href="#">en-pos-perceptron.bin</a>
en	Name Finder	Date name finder model.	<a href="#">en-ner-date.bin</a>
en	Name Finder	Location name finder model.	<a href="#">en-ner-location.bin</a>
en	Name Finder	Money name finder model.	<a href="#">en-ner-money.bin</a>
en	Name Finder	Organization name finder model.	<a href="#">en-ner-organization.bin</a>
en	Name Finder	Percentage name finder model.	<a href="#">en-ner-percentage.bin</a>
en	Name Finder	Person name finder model.	<a href="#">en-ner-person.bin</a>
en	Name Finder	Time name finder model.	<a href="#">en-ner-time.bin</a>
en	Chunker	Trained on conll2000 shared task data.	<a href="#">en-chunker.bin</a>
en	Parser		<a href="#">en-parser-chunking.bin</a>
en	Coreference		<a href="#">coref</a>

Download all these models to the folder **C:/OpenNLP\_models/**, by clicking on their respective links. All these models are language dependent and while using these, you have to make sure that the model language matches with the language of the input text.



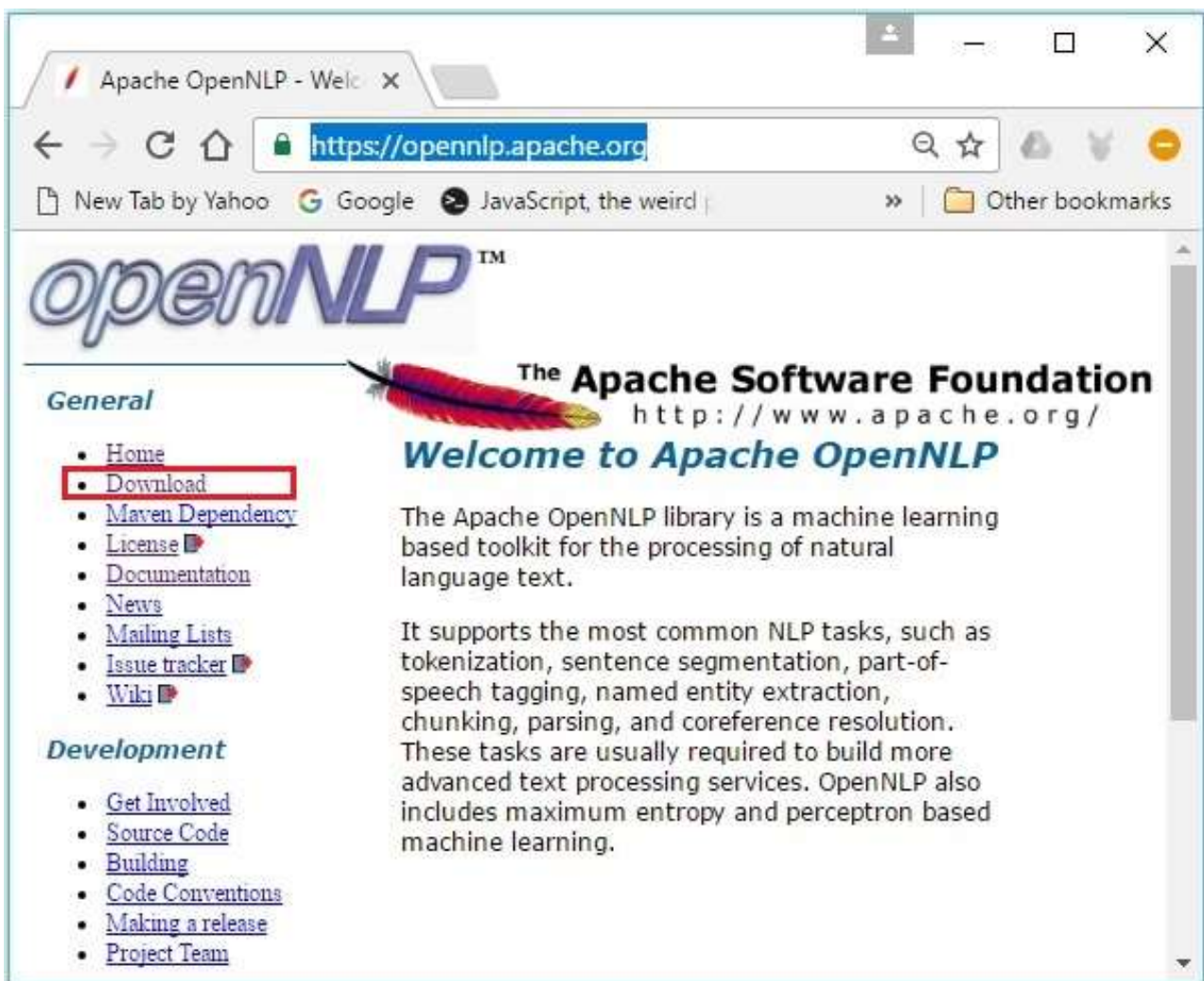
## 2. OpenNLP – Environment

In this chapter, we will discuss how you can setup OpenNLP environment in your system. Let's start with the installation process.

### Installing OpenNLP

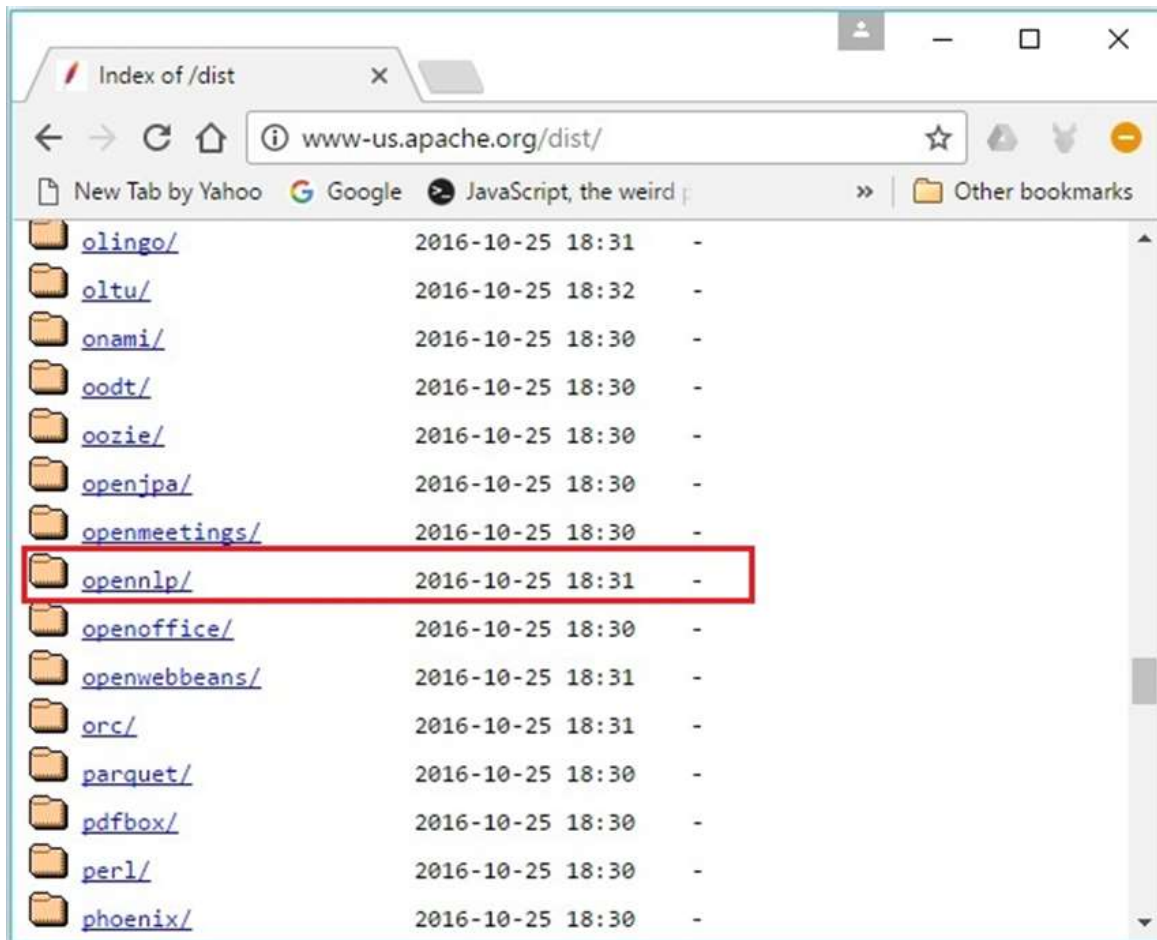
Following are the steps to download **Apache OpenNLP library** in your system.

**Step 1:** Open the homepage of **Apache OpenNLP** by clicking the following link: <https://opennlp.apache.org/>

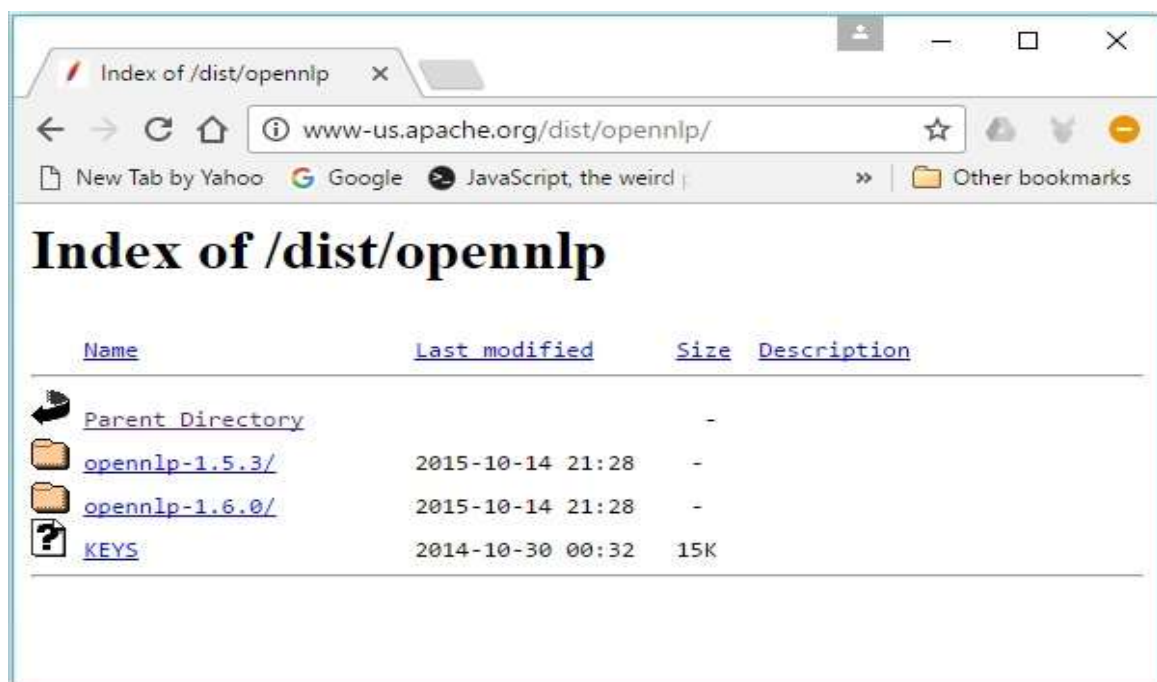


**Step 2:** Now, click on the **Downloads** link. On clicking, you will be directed to a page where you can find various mirrors which will redirect you to the Apache Software Foundation Distribution directory.

**Step 3:** In this page you can find links to download various Apache distributions. Browse through them and find the OpenNLP distribution and click it.

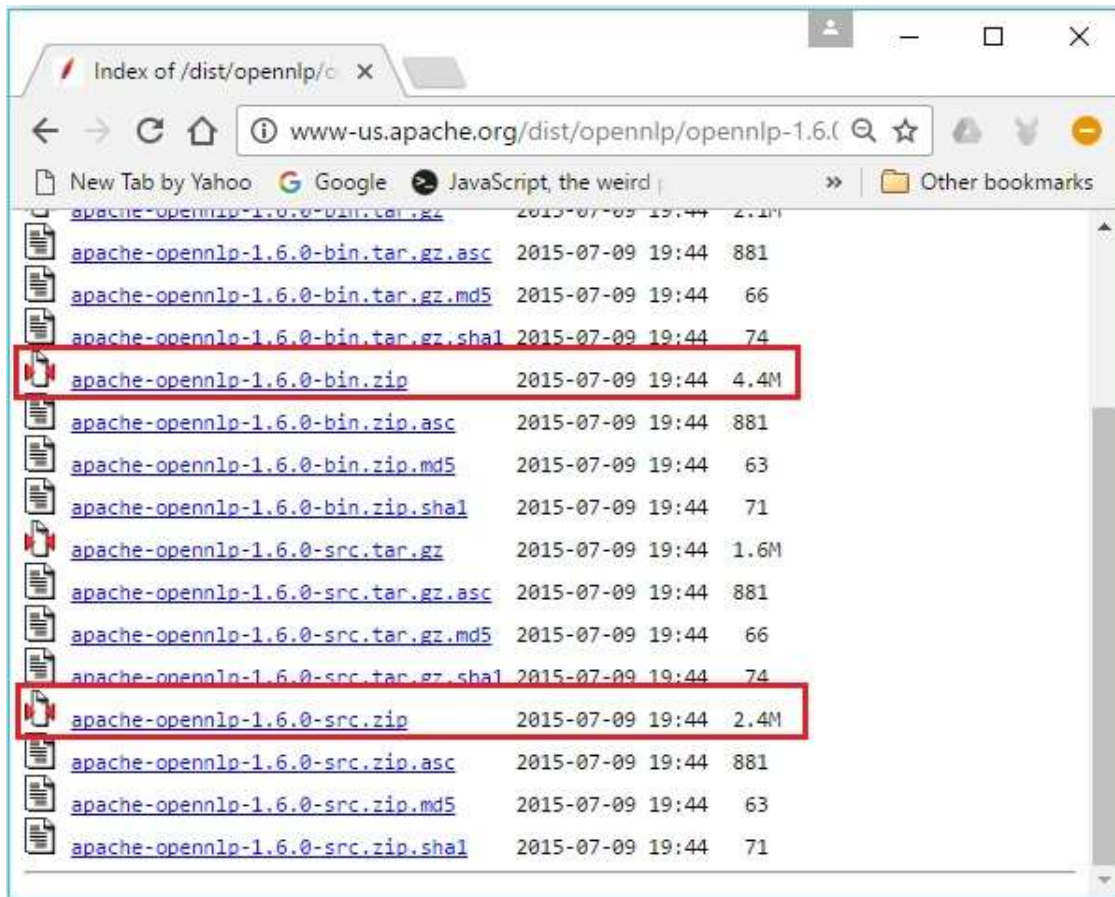


**Step 4:** On clicking, you will be redirected to the directory where you can see the index of the OpenNLP distribution, as shown below.



Click on the latest version from the available distributions.

**Step 5:** Each distribution provides Source and Binary files of OpenNLP library in various formats. Download the source and binary files, **apache-opennlp-1.6.0-bin.zip** and **apache-opennlp-1.6.0-src.zip** (for Windows).



## Setting the Classpath

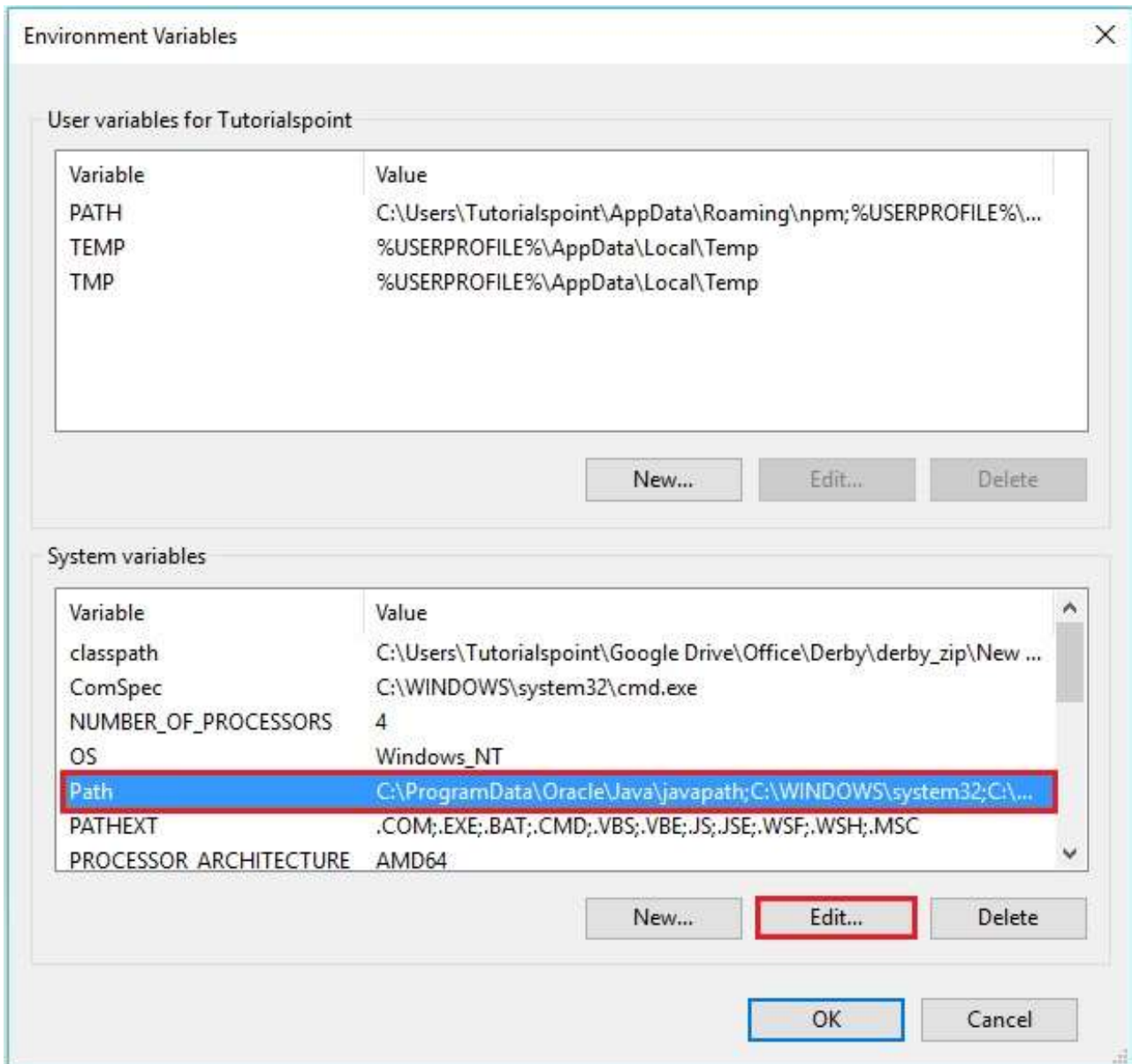
After downloading the OpenNLP library, you need to set its path to the **bin** directory. Assume that you have downloaded the OpenNLP library to the E drive of your system.

Now, follow the steps that are given below:

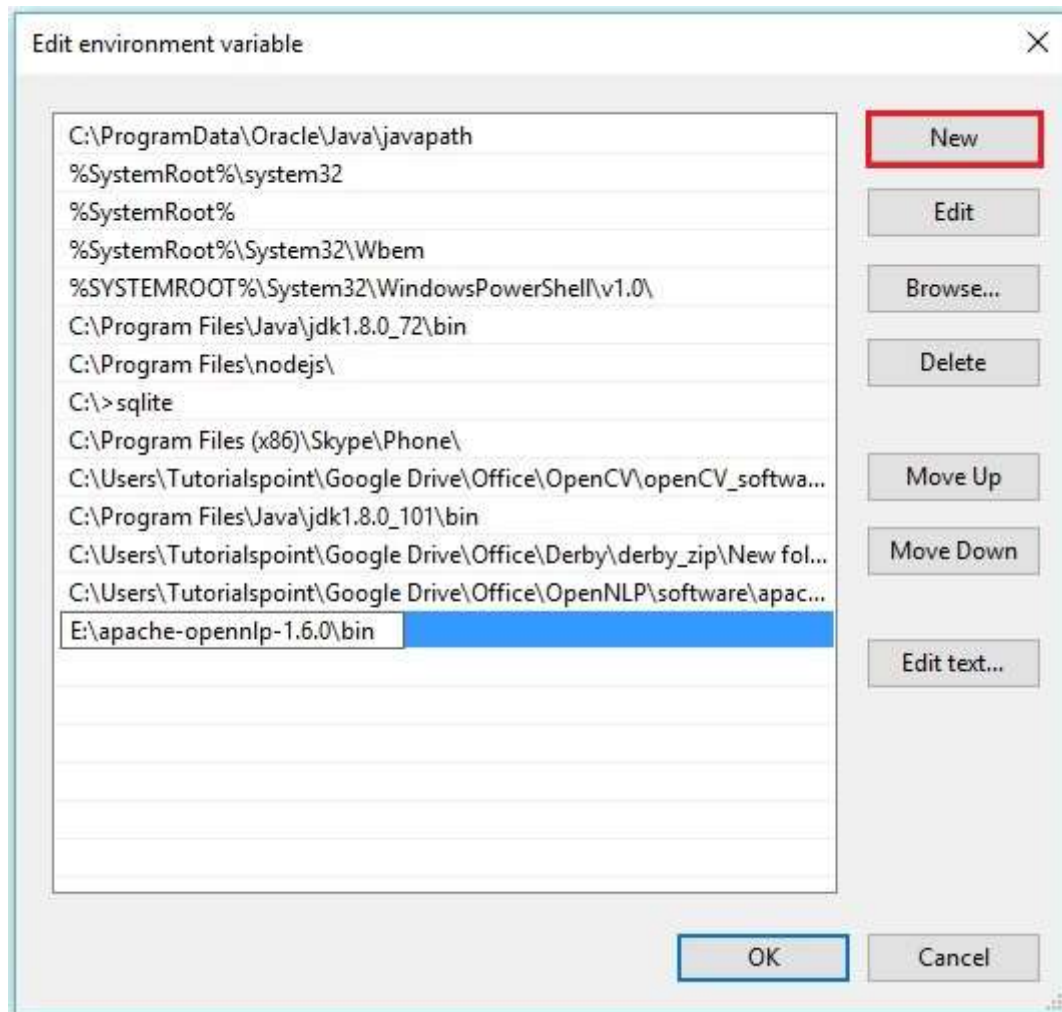
**Step 1:** Right-click on 'My Computer' and select 'Properties'.

**Step 2:** Click on the 'Environment Variables' button under the 'Advanced' tab.

**Step 3:** Select the **path** variable and click the **Edit** button, as shown in the following screenshot.



**Step 4:** In the *Edit Environment Variable* window, click the **New** button and add the path for OpenNLP directory **E:\apache-opennlp-1.6.0\bin** and click the **OK** button, as shown in the following screenshot.



## Eclipse Installation

You can set the Eclipse environment for OpenNLP library, either by setting the **Build path** to the JAR files or by using **pom.xml**.

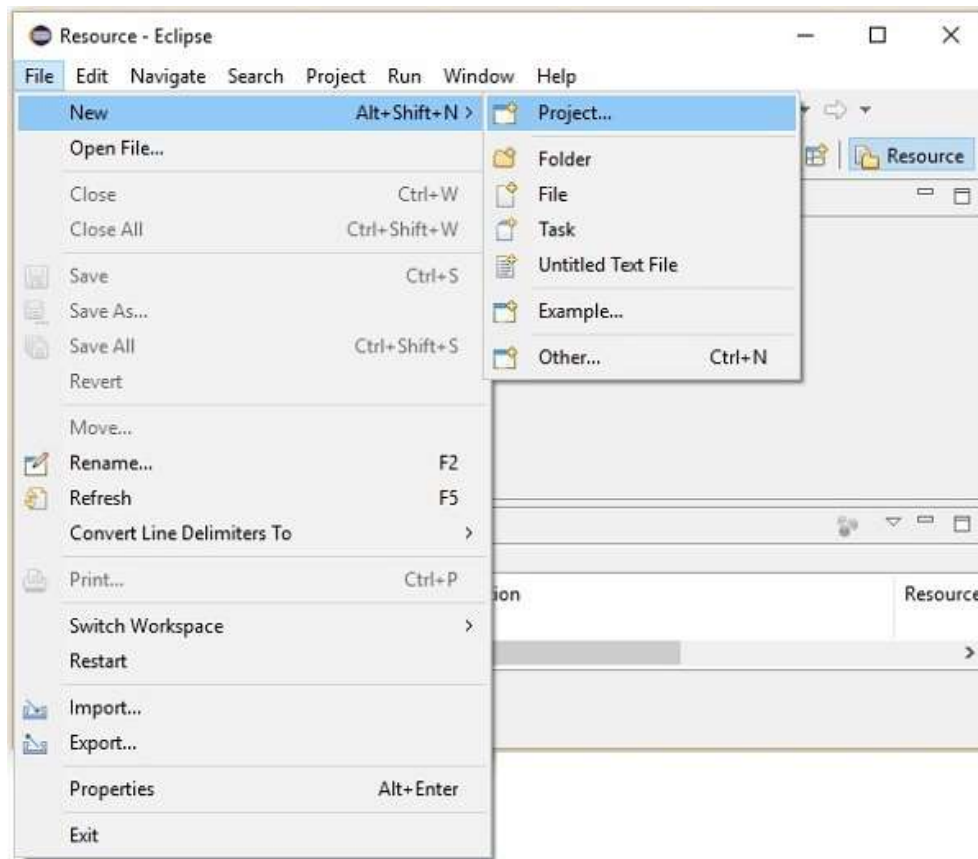
### Setting Build Path to the JAR Files

Follow the steps given below to install OpenNLP in Eclipse:

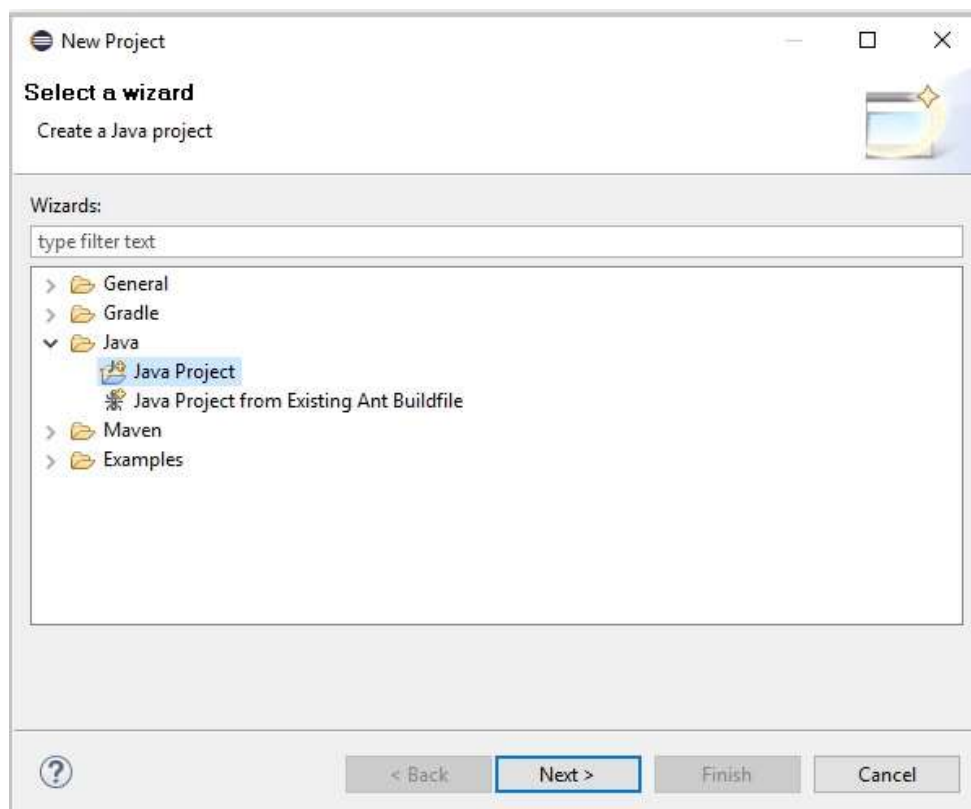
**Step 1:** Make sure that you have Eclipse environment installed in your system.

**Step 2:** Open Eclipse. Click File -> New -> Open a new project, as shown below.





**Step 3:** You will get the **New Project** wizard. In this wizard, select Java project and proceed by clicking the **Next** button.



**Step 4:** Next, you will get the **New Java Project wizard**. Here, you need to create a new project and click the **Next** button, as shown below.

New Java Project

### Create a Java Project

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location:  [Browse...](#)

JRE

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE (currently 'jre1.8.0\_72') [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

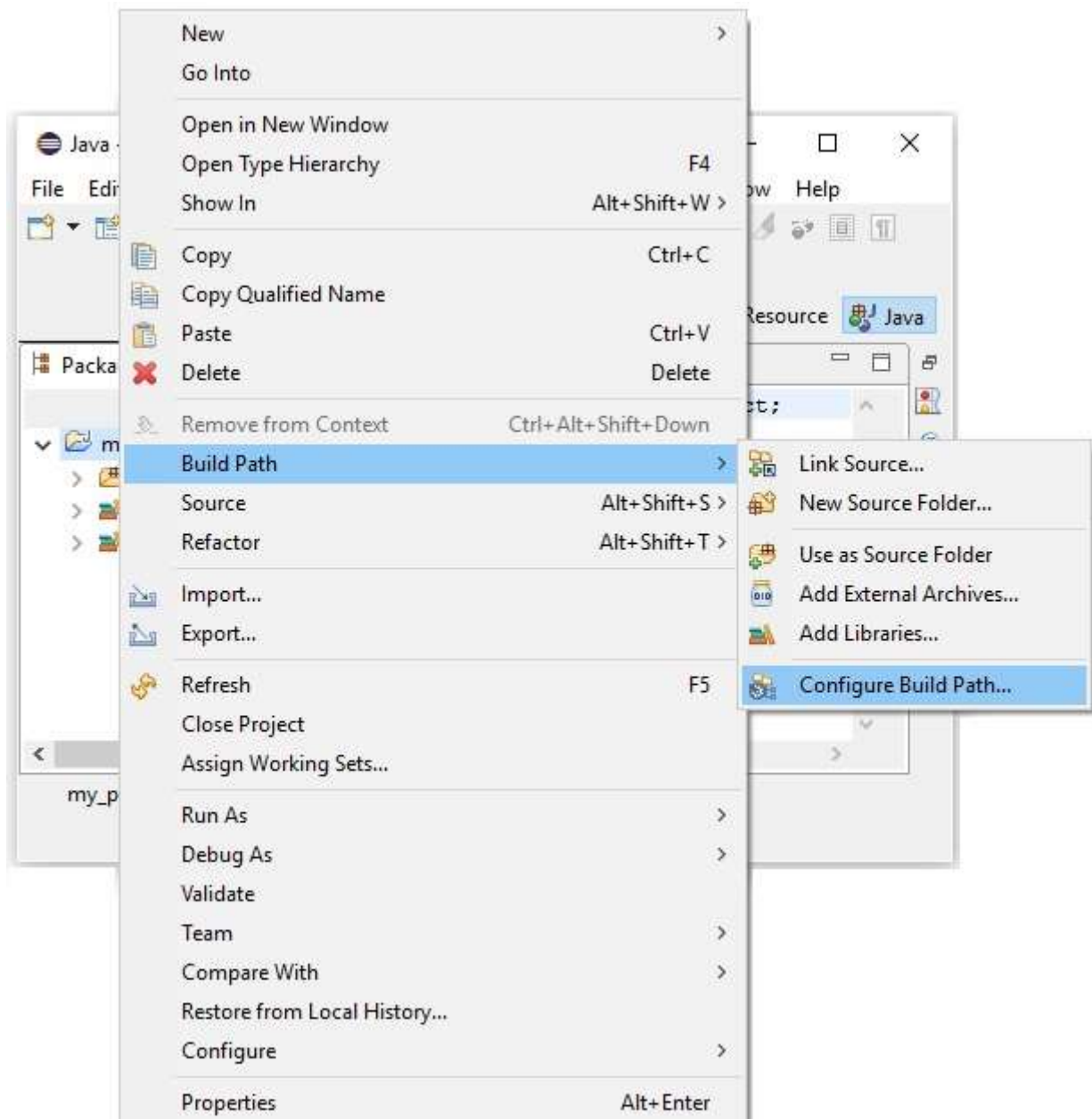
Working sets

☐ Add project to working sets

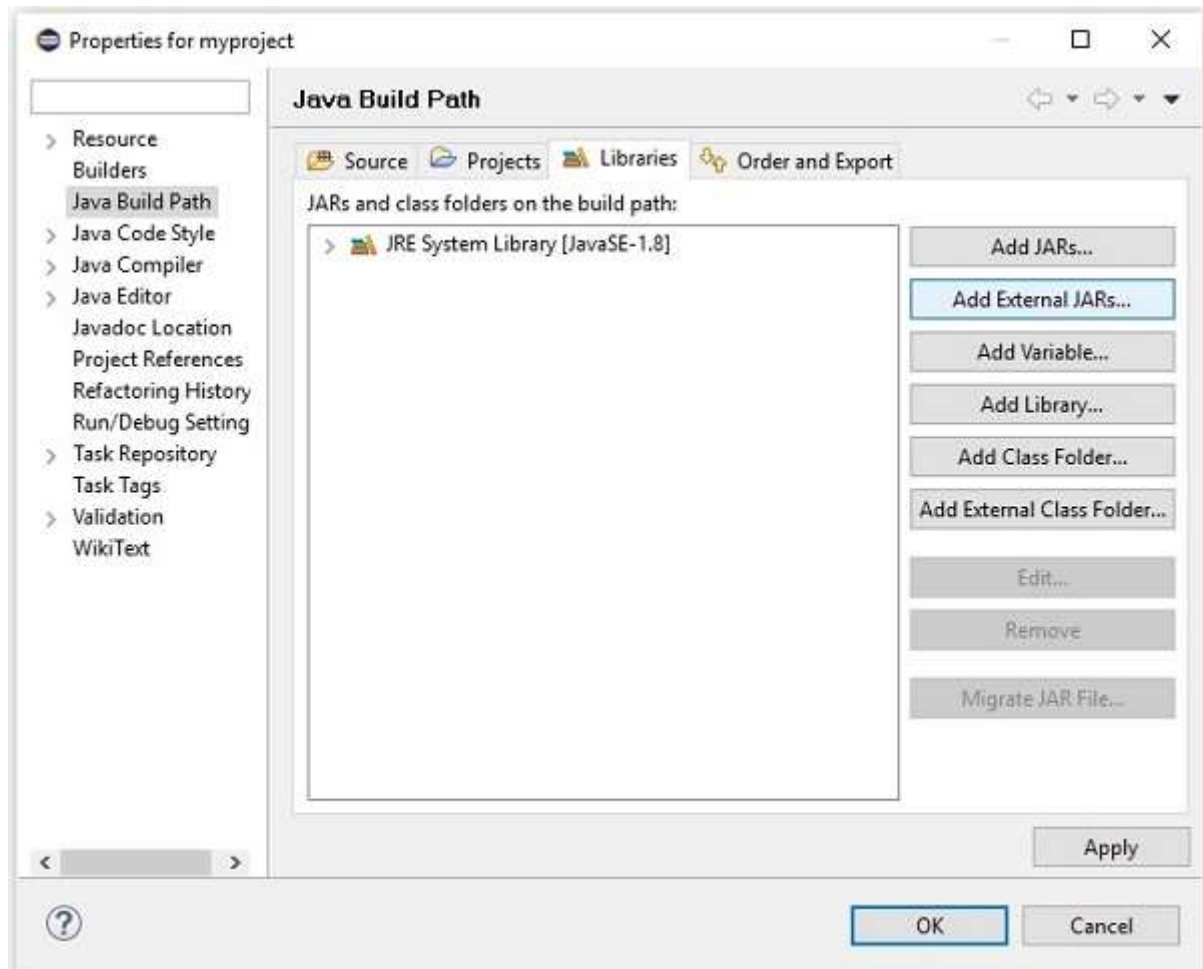
Working sets:  [Select...](#)

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

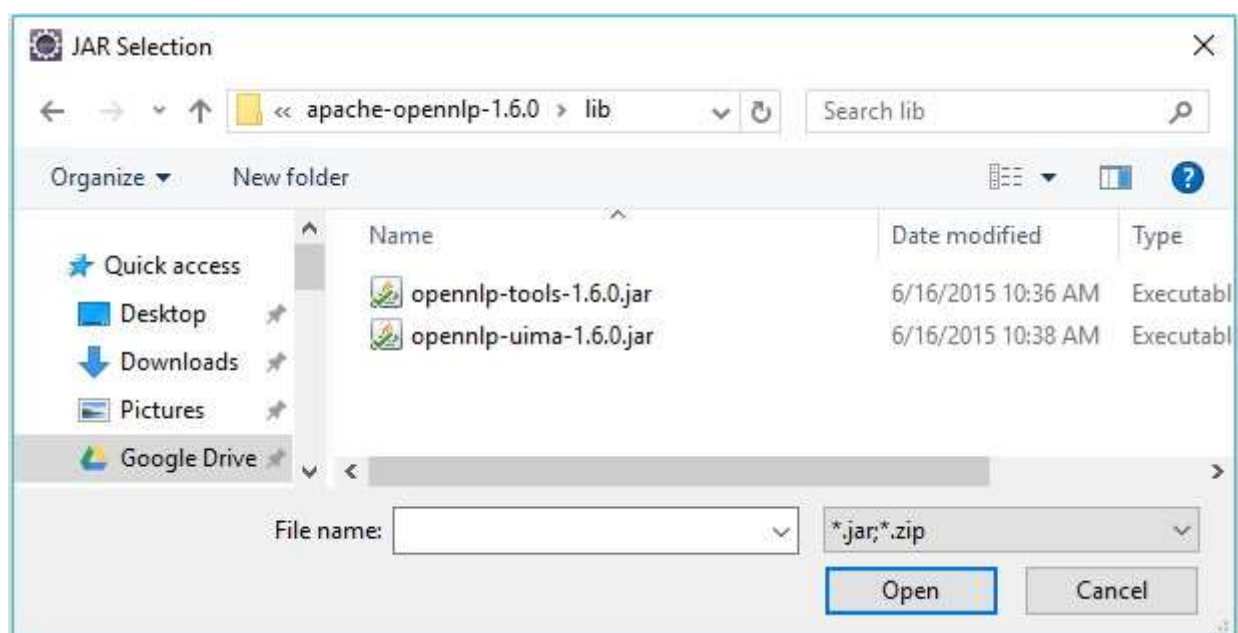
**Step 5:** After creating a new project, right-click on it, select **Build Path** and click **Configure Build Path**.



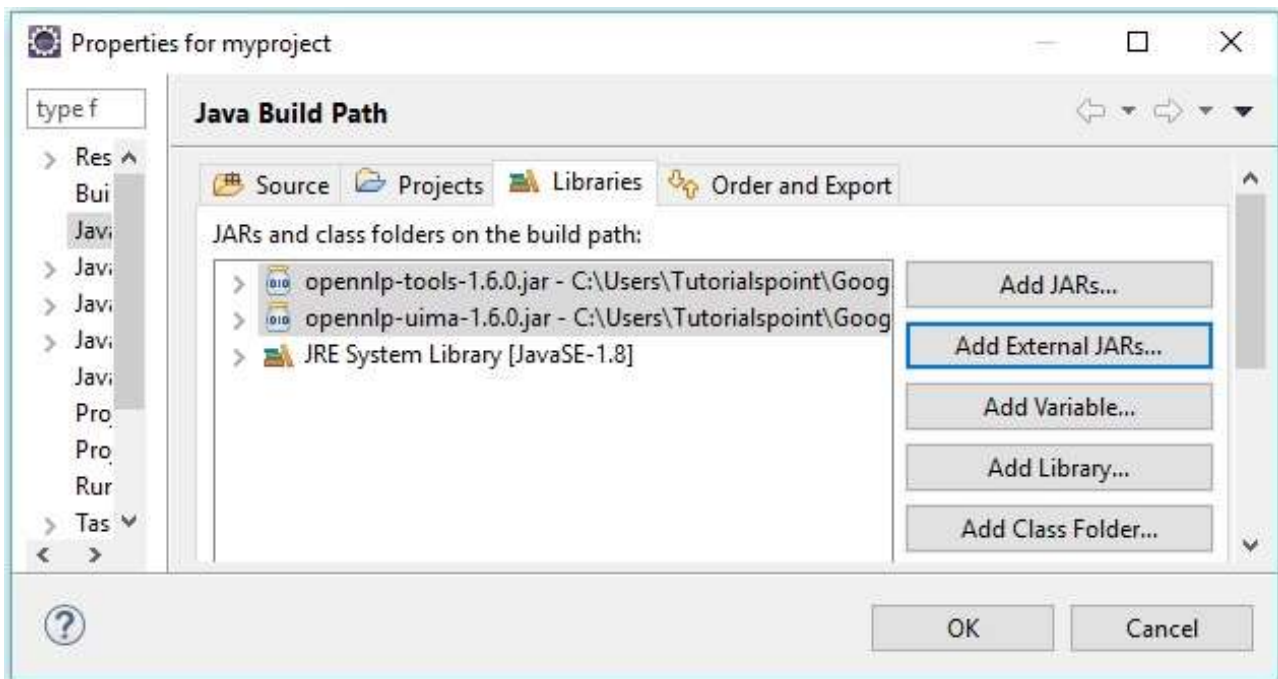
**Step 6:** Next, you will get the **Java Build Path** wizard. Here, click the **Add External JARs** button, as shown below.



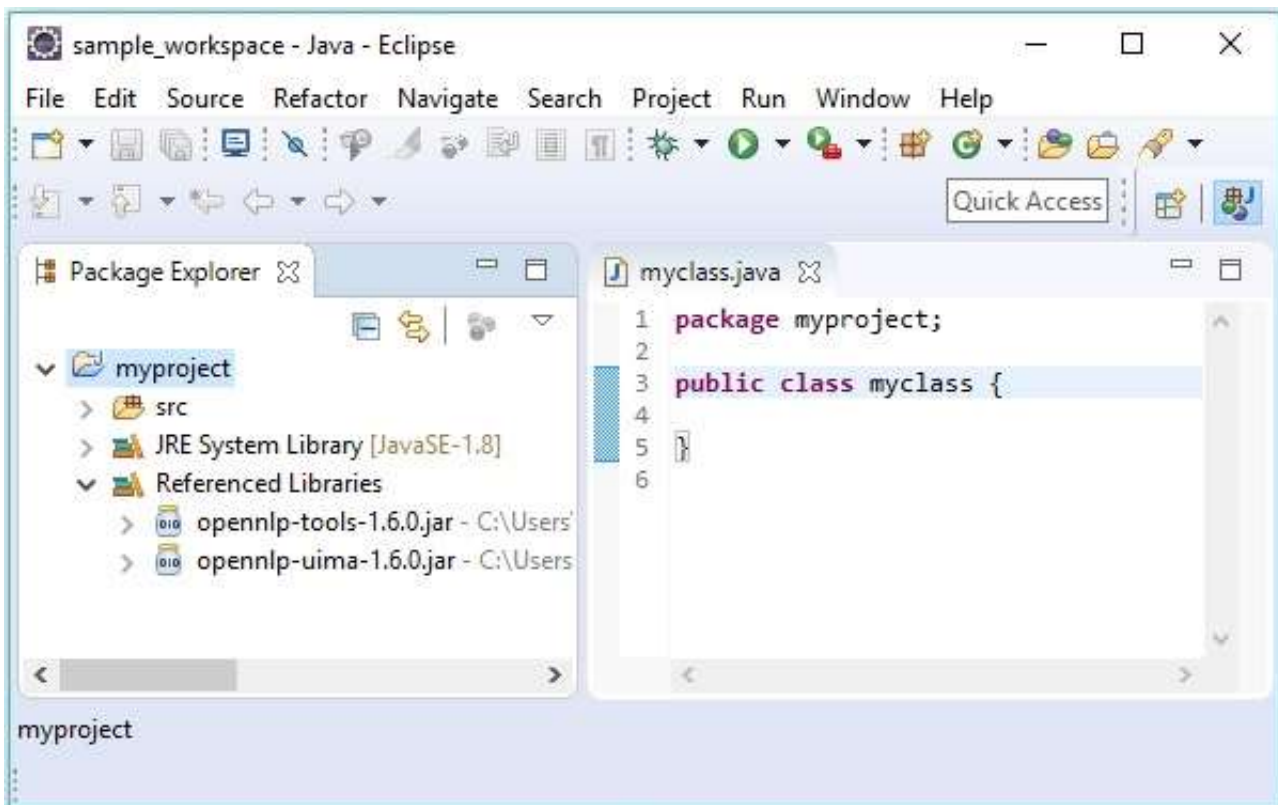
**Step 7:** Select the jar files **opennlp-tools-1.6.0.jar** and **opennlp-uima-1.6.0.jar** located in the **lib** folder of **apache-opennlp-1.6.0** folder.



On clicking the **Open** button in the above screen, the selected files will be added to your library.



On clicking **OK**, you will successfully add the required JAR files to the current project and you can verify these added libraries by expanding the Referenced Libraries, as shown below.





## Using pom.xml

Convert the project into a Maven project and add the following code to its **pom.xml**.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>myproject</groupId>
  <artifactId>myproject</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>org.apache.opennlp</groupId>
      <artifactId>opennlp-tools</artifactId>
      <version>1.6.0</version>
    </dependency>
    <dependency>
      <groupId>org.apache.opennlp</groupId>
      <artifactId>opennlp-uima</artifactId>
      <version>1.6.0</version>
    </dependency>
  </dependencies>
</project>
```

## 3. OpenNLP – Referenced API

In this chapter, we will discuss about the classes and methods that we will be using in the subsequent chapters of this tutorial.

### Sentence Detection

---

#### SentenceModel

This class represents the predefined model which is used to detect the sentences in the given raw text. This class belongs to the package **opennlp.tools.sentence**.

The constructor of this class accepts an **InputStream** object of the sentence detector model file (en-sent.bin).

#### SentenceDetectorME class

This class belongs to the package **opennlp.tools.sentence** and it contains methods to split the raw text into sentences. This class uses a maximum entropy model to evaluate end-of-sentence characters in a string to determine if they signify the end of a sentence.

S.No	Methods and Description
1	<b>sentDetect()</b> This method is used to detect the sentences in the raw text passed to it. It accepts a String variable as a parameter and returns a String array which holds the sentences from the given raw text.
2	<b>sentPosDetect()</b> This method is used to detect the positions of the sentences in the given text. This method accepts a string variable, representing the sentence and returns an array of objects of the type <b>Span</b> .  The class named <b>Span</b> of the <b>opennlp.tools.util</b> package is used to store the start and end integer of sets.
3	<b>getSentenceProbabilities()</b> This method returns the probabilities associated with the most recent calls to <b>sentDetect()</b> method.

## Tokenization

---

### TokenizerModel

This class represents the predefined model which is used to tokenize the given sentence. This class belongs to the package **opennlp.tools.tokenizer**.

The constructor of this class accepts a **InputStream** object of the tokenizer model file (en-token.bin).

### Classes

To perform tokenization, the OpenNLP library provides three main classes. All the three classes implement the interface called **Tokenizer**.

S.No	Classes and Description
1	<b>SimpleTokenizer</b> This class tokenizes the given raw text using character classes.
2	<b>WhitespaceTokenizer</b> This class uses whitespaces to tokenize the given text.
3	<b>TokenizerME</b> This class converts raw text in to separate tokens. It uses Maximum Entropy to make its decisions.

These classes contain the following methods.

S.No	Methods and Description
1	<b>tokenize()</b> This method is used to tokenize the raw text. This method accepts a String variable as a parameter, and returns an array of Strings (tokens).
2	<b>sentPosDetect()</b> This method is used to get the positions or spans of the tokens. It accepts the sentence (or) raw text in the form of the string and returns an array of objects of the type <b>Span</b> .

In addition to the above two methods, the **TokenizerME** class has the **getTokenProbabilities()** method.

S.No	Methods and Description
1	<b>getTokenProbabilities()</b> This method is used to get the probabilities associated with the most recent calls to the <b>tokenizePos()</b> method.

## NameEntityRecognition

---

### TokenNameFinderModel

This class represents the predefined model which is used to find the named entities in the given sentence. This class belongs to the package **opennlp.tools.namefind**.

The constructor of this class accepts a **InputStream** object of the name finder model file (en-ner-person.bin).

### NameFinderME class

The class belongs to the package **opennlp.tools.namefind** and it contains methods to perform the NER tasks. This class uses a maximum entropy model to find the named entities in the given raw text.

S.No	Methods and Description
1	<b>find()</b> This method is used to detect the names in the raw text. It accepts a String variable representing the raw text as a parameter and, returns an array of objects of the type Span.
2	<b>probs()</b> This method is used to get the probabilities of the last decoded sequence.

## Finding the Parts of Speech

---

### POSModel

This class represents the predefined model which is used to tag the parts of speech of the given sentence. This class belongs to the package **opennlp.tools.postag**.

The constructor of this class accepts a **InputStream** object of the pos-tagger model file (en-pos-maxent.bin).

## POSTaggerME class

This class belongs to the package **opennlp.tools.postag** and it is used to predict the parts of speech of the given raw text. It uses Maximum Entropy to make its decisions.

S.No	Methods and Description
1	<b>tag()</b> This method is used to assign the sentence of tokens POS tags. This method accepts an array of tokens (String) as a parameter, and returns a tags (array).
2	<b>getSentenceProbabilities()</b> This method is used to get the probabilities for each tag of the recently tagged sentence.

## Parsing the Sentence

### ParserModel

This class represents the predefined model which is used to parse the given sentence. This class belongs to the package **opennlp.tools.parser**.

The constructor of this class accepts a **InputStream** object of the parser model file (en-parser-chunking.bin).

### Parser Factory class

This class belongs to the package **opennlp.tools.parser** and it is used to create parsers.

S.No	Methods and Description
1	<b>create ()</b> This is a static method and it is used to create a parser object. This method accepts the Filestream object of the parser model file.



## ParserTool class

This class belongs to the **opennlp.tools.cmdline.parser** package and, it is used to parse the content.

S.No	Methods and Description
1	<p><b>parseLine()</b></p> <p>This method of the <b>ParserTool</b> class is used to parse the raw text in OpenNLP. This method accepts –</p> <ul style="list-style-type: none"> <li>• A String variable representing the text to be parsed.</li> <li>• A parser object.</li> <li>• An integer representing the no.of parses to be carried out.</li> </ul>

## Chunking

### ChunkerModel

This class represents the predefined model which is used to divide a sentence into smaller chunks. This class belongs to the package **opennlp.tools.chunker**.

The constructor of this class accepts a **InputStream** object of the **chunker** model file (en-chunker.bin).

### ChunkerME class

This class belongs to the package named **opennlp.tools.chunker** and it is used to divide the given sentence in to smaller chunks.

S.No	Methods and Description
1	<p><b>chunk()</b></p> <p>This method is used to divide the given sentence in to smaller chunks. It accepts tokens of a sentence and <b>Parts Of Speech</b> tags as parameters.</p>
2	<p><b>probs()</b></p> <p>This method returns the probabilities of the last decoded sequence.</p>

## 4. Open NLP – Sentence Detection

While processing a natural language, deciding the beginning and end of the sentences is one of the problems to be addressed.

This process is known as **S**entence **B**oundary **D**isambiguation (SBD) or simply sentence breaking.

The techniques we use to detect the sentences in the given text depends on the language of the text.

### Sentence Detection Using Java

---

We can detect the sentences in the given text in Java using, Regular Expressions, and a set of simple rules.

For example, let us assume a period, a question mark, or an exclamation mark ends a sentence in the given text, then we can split the sentence using the **split()** method of the **String** class. Here, we have to pass a regular expression in String format.

Following is the program which determines the sentences in a given text using Java regular expressions (**split method**). Save this program in a file with the name **SentenceDetection\_RE.java**.

```
public class SentenceDetection_RE {  
  
    public static void main(String args[]){  
  
        String sentence = " Hi. How are you? Welcome to Tutorialspoint. "  
            + "We provide free tutorials on various technologies";  
  
        String simple = "[.?!]";  
        String[] splitString = (sentence.split(simple));  
        for (String string : splitString)  
            System.out.println(string);  
    }  
}
```

Compile and execute the saved java file from the command prompt using the following commands

```
javac SentenceDetection_RE.java  
java SentenceDetection_RE
```

On executing, the above program creates a PDF document displaying the following message.

```
Hi
How are you
Welcome to Tutorialspoint
We provide free tutorials on various technologies
```

## Sentence Detection Using OpenNLP

To detect sentences, OpenNLP uses a predefined model, a file named **en-sent.bin**. This predefined model is trained to detect sentences in a given raw text.

The **opennlp.tools.sntdetect** package contains the classes and interfaces that are used to perform the sentence detection task.

To detect a sentence using OpenNLP library, you need to –

- Load the **en-sent.bin** model using the **SentenceModel** class
- Instantiate the **SentenceDetectorME** class
- Detect the sentences using the **sentDetect()** method of this class.

Following are the steps to be followed to write a program which detects the sentences from the given raw text.

### Step 1: Loading the model

The model for sentence detection is represented by the class named **SentenceModel**, which belongs to the package **opennlp.tools.sntdetect**.

To load a sentence detection model –

- Create an **InputStream** object of the model (Instantiate the **FileInputStream** and pass the path of the model in String format to its constructor).
- Instantiate the **SentenceModel** class and pass the **InputStream** (object) of the model as a parameter to its constructor as shown in the following code block –

```
//Loading sentence detector model
InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-
sent.bin");
SentenceModel model = new SentenceModel(inputStream);
```

### Step 2: Instantiating the SentenceDetectorME class

The **SentenceDetectorME** class of the package **opennlp.tools.sntdetect** contains methods to split the raw text into sentences. This class uses the Maximum Entropy model to evaluate end-of-sentence characters in a string to determine if they signify the end of a sentence.

Instantiate this class and pass the model object created in the previous step, as shown below.

```
//Instantiating the SentenceDetectorME class  
SentenceDetectorME detector = new SentenceDetectorME(model);
```

### Step 3: Detecting the sentence

The **sentDetect()** method of the **SentenceDetectorME** class is used to detect the sentences in the raw text passed to it. This method accepts a String variable as a parameter.

Invoke this method by passing the String format of the sentence to this method.

```
//Detecting the sentence  
String sentences[] = detector.sentDetect(sentence);
```

### Example

Following is the program which detects the sentences in a given raw text. Save this program in a file with named **SentenceDetectionME.java**.

```
import java.io.FileInputStream;  
import java.io.InputStream;  
  
import opennlp.tools.sentence.SentenceDetectorME;  
import opennlp.tools.sentence.SentenceModel;  
  
public class SentenceDetectionME {  
  
    public static void main(String args[]) throws Exception {  
  
        String sentence = "Hi. How are you? Welcome to Tutorialspoint. "  
            + "We provide free tutorials on various technologies";  
  
        //Loading sentence detector model  
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-sent.bin");  
        SentenceModel model = new SentenceModel(inputStream);  
  
        //Instantiating the SentenceDetectorME class  
        SentenceDetectorME detector = new SentenceDetectorME(model);  
  
        //Detecting the sentence
```

```

String sentences[] = detector.sentDetect(sentence);

//Printing the sentences
for(String sent : sentences)
    System.out.println(sent);
}
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```

javac SentenceDetectorME.java
java SentenceDetectorME

```

On executing, the above program reads the given String and detects the sentences in it and displays the following output.

```

Hi. How are you?
Welcome to Tutorialspoint.
We provide free tutorials on various technologies

```

## Detecting the Positions of the Sentences

We can also detect the positions of the sentences using the **sentPosDetect()** method of the **SentenceDetectorME** class.

Following are the steps to be followed to write a program which detects the positions of the sentences from the given raw text.

### Step 1: Loading the model

The model for sentence detection is represented by the class named **SentenceModel**, which belongs to the package **opennlp.tools.sntdetect**.

To load a sentence detection model –

- Create an **InputStream** object of the model (Instantiate the **FileInputStream** and pass the path of the model in String format to its constructor).
- Instantiate the **SentenceModel** class and pass the **InputStream** (object) of the model as a parameter to its constructor, as shown in the following code block –

```

//Loading sentence detector model
InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-sent.bin");
SentenceModel model = new SentenceModel(inputStream);

```



## Step 2: Instantiating the SentenceDetectorME class

The **SentenceDetectorME** class of the package **opennlp.tools.sntdetect** contains methods to split the raw text into sentences. This class uses the Maximum Entropy model to evaluate end-of-sentence characters in a string to determine if they signify the end of a sentence.

Instantiate this class and pass the model object created in the previous step.

```
//Instantiating the SentenceDetectorME class  
SentenceDetectorME detector = new SentenceDetectorME(model);
```

## Step 3: Detecting the position of the sentence

The **sentPosDetect()** method of the **SentenceDetectorME** class is used to detect the positions of the sentences in the raw text passed to it. This method accepts a String variable as a parameter.

Invoke this method by passing the String format of the sentence as a parameter to this method.

```
//Detecting the position of the sentences in the paragraph  
Span[] spans = detector.sentPosDetect(sentence);
```

## Step 4: Printing the spans of the sentences

The **sentPosDetect()** method of the **SentenceDetectorME** class returns an array of objects of the type **Span**. The class named Span of the **opennlp.tools.util** package is used to store the start and end integer of sets.

You can store the spans returned by the **sentPosDetect()** method in the Span array and print them, as shown in the following code block.

```
//Printing the sentences and their spans of a sentence  
for (Span span : spans)  
System.out.println(paragraph.substring(span);
```

## Example

Following is the program which detects the sentences in the given raw text. Save this program in a file with named **SentenceDetectionME.java**.

```
import java.io.FileInputStream;  
import java.io.InputStream;  
  
import opennlp.tools.sntdetect.SentenceDetectorME;  
import opennlp.tools.sntdetect.SentenceModel;  
import opennlp.tools.util.Span;
```

```
public class SentencePosDetection {  
  
    public static void main(String args[]) throws Exception {  
  
        String paragraph = "Hi. How are you? Welcome to Tutorialspoint. "  
            + "We provide free tutorials on various technologies";  
  
        //Loading sentence detector model  
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-sent.bin");  
        SentenceModel model = new SentenceModel(inputStream);  
  
        //Instantiating the SentenceDetectorME class  
        SentenceDetectorME detector = new SentenceDetectorME(model);  
  
        //Detecting the position of the sentences in the raw text  
        Span spans[] = detector.sentPosDetect(paragraph);  
  
        //Printing the spans of the sentences in the paragraph  
        for (Span span : spans)  
            System.out.println(span);  
    }  
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac SentencePosDetection.java  
java SentencePosDetection
```

On executing, the above program reads the given String and detects the sentences in it and displays the following output.

```
[0..16)  
[17..43)  
[44..93)
```

## Sentences along with their Positions

The **substring()** method of the String class accepts the **begin** and the **end offsets** and returns the respective string. We can use this method to print the sentences and their spans (positions) together, as shown in the following code block.

```
for (Span span : spans)
    System.out.println(sen.substring(span.getStart(), span.getEnd())+" "+ span);
```

Following is the program to detect the sentences from the given raw text and display them along with their positions. Save this program in a file with name **SentencesAndPosDetection.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;

import opennlp.tools.sntdetect.SentenceDetectorME;
import opennlp.tools.sntdetect.SentenceModel;
import opennlp.tools.util.Span;

public class SentencesAndPosDetection {

    public static void main(String args[]) throws Exception {

        String sen = "Hi. How are you? Welcome to Tutorialspoint."
            + " We provide free tutorials on various technologies";
        //Loading a sentence model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-sent.bin");
        SentenceModel model = new SentenceModel(inputStream);

        //Instantiating the SentenceDetectorME class
        SentenceDetectorME detector = new SentenceDetectorME(model);

        //Detecting the position of the sentences in the paragraph
        Span[] spans = detector.sentPosDetect(sen);

        //Printing the sentences and their spans of a paragraph
        for (Span span : spans)
            System.out.println(sen.substring(span.getStart(), span.getEnd())+" "+ span);
    }
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac SentencesAndPosDetection.java
java SentencesAndPosDetection
```

On executing, the above program reads the given String and detects the sentences along with their positions and displays the following output.

```
Hi. How are you? [0..16)
Welcome to Tutorialspoint. [17..43)
We provide free tutorials on various technologies [44..93)
```

## Sentence Probability Detection

The **getSentenceProbabilities()** method of the **SentenceDetectorME** class returns the probabilities associated with the most recent calls to the **sentDetect()** method.

```
//Getting the probabilities of the last decoded sequence
double[] probs = detector.getSentenceProbabilities();
```

Following is the program to print the probabilities associated with the calls to the **sentDetect()** method. Save this program in a file with the name **SentenceDetectionMEProbs.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;

import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;

public class SentenceDetectionMEProbs {

    public static void main(String args[]) throws Exception {

        String sentence = "Hi. How are you? Welcome to Tutorialspoint. "
            + "We provide free tutorials on various technologies";

        //Loading sentence detector model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-sent.bin");
        SentenceModel model = new SentenceModel(inputStream);

        //Instantiating the SentenceDetectorME class
```

```

SentenceDetectorME detector = new SentenceDetectorME(model);

//Detecting the sentence
String sentences[] = detector.sentDetect(sentence);

//Printing the sentences
for(String sent : sentences)
    System.out.println(sent);

//Getting the probabilities of the last decoded sequence
double[] probs = detector.getSentenceProbabilities();

System.out.println(" ");

for(int i=0; i<probs.length; i++)
    System.out.println(probs[i]);
}
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```

javac SentenceDetectionMEProbs.java
java SentenceDetectionMEProbs

```

On executing, the above program reads the given String and detects the sentences and prints them. In addition, it also returns the probabilities associated with the most recent calls to the `sentDetect()` method, as shown below.

```

Hi. How are you?
Welcome to Tutorialspoint.
We provide free tutorials on various technologies

0.9240246995179983
0.9957680129995953
1.0

```

## 5. Open NLP – Tokenization

The process of chopping the given sentence into smaller parts (tokens) is known as **tokenization**. In general, the given raw text is tokenized based on a set of delimiters (mostly whitespaces).

Tokenization is used in tasks such as spell-checking, processing searches, identifying parts of speech, sentence detection, document classification of documents, etc.

### Tokenizing using OpenNLP

---

The **opennlp.tools.tokenize** package contains the classes and interfaces that are used to perform tokenization.

To tokenize the given sentences into simpler fragments, the OpenNLP library provides three different classes:

- **SimpleTokenizer**: This class tokenizes the given raw text using character classes.
- **WhitespaceTokenizer**: This class uses whitespaces to tokenize the given text.
- **TokenizerME**: This class converts raw text into separate tokens. It uses Maximum Entropy to make its decisions.

### SimpleTokenizer and WhitespaceTokenizer classes

To tokenize a sentence using the classes **SimpleTokenizer** and **TokenizerME**, you need to –

- Create an object of the respective class.
- Tokenize the sentence using the **tokenize()** method.
- Print the tokens.

Following are the steps to be followed to write a program which tokenizes the given raw text.

### Step 1: Instantiating the respective class

In both the classes, there are no constructors available to instantiate them. Therefore, we need to create objects of these classes using the static variable **INSTANCE**.

```
SimpleTokenizer tokenizer = SimpleTokenizer.INSTANCE;
```

Or

```
WhitespaceTokenizer tokenizer = WhitespaceTokenizer.INSTANCE;
```



## Step 2: Tokenize the sentences

Both these classes contain a method called **tokenize()**. This method accepts a raw text in String format. On invoking, it tokenizes the given String and returns an array of Strings (tokens).

Tokenize the sentence using the **tokenizer()** method as shown below.

```
//Tokenizing the given sentence
String tokens[] = tokenizer.tokenize(sentence);
```

## Step 3: Print the tokens

After tokenizing the sentence, you can print the tokens using **for loop**, as shown below.

```
//Printing the tokens
for(String token : tokens)
    System.out.println(token);
```

## SimpleTokenizer example

Following is the program which tokenizes the given sentence using the SimpleTokenizer class. Save this program in a file with the name **SimpleTokenizerExample.java**.

```
import opennlp.tools.tokenize.SimpleTokenizer;

public class SimpleTokenizerExample {
    public static void main(String args[]){

        String sentence = "Hi. How are you? Welcome to Tutorialspoint. "
            + "We provide free tutorials on various technologies";

        //Instantiating SimpleTokenizer class
        SimpleTokenizer simpleTokenizer = SimpleTokenizer.INSTANCE;

        //Tokenizing the given sentence
        String tokens[] =simpleTokenizer.tokenize(sentence);

        //Printing the tokens
        for(String token : tokens) {
            System.out.println(token);
        }
    }
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac SimpleTokenizerExample.java
java SimpleTokenizerExample
```

On executing, the above program reads the given String (raw text), tokenizes it, and displays the following output:

```
Hi
.
How
are
you
?
Welcome
to
Tutorialspoint
.
We
provide
free
tutorials
on
various
technologies
```

## WhitespaceTokenizer example

Following is the program which tokenizes the given sentence using the **WhitespaceTokenizer** class. Save this program in a file with the name **WhitespaceTokenizerExample.java**.

```
import opennlp.tools.tokenize.WhitespaceTokenizer;

public class WhitespaceTokenizerExample {

    public static void main(String args[]){

        String sentence = "Hi. How are you? Welcome to Tutorialspoint. "
            + "We provide free tutorials on various technologies";

        //Instantiating whitespaceTokenizer class
```

```

WhitespaceTokenizer whitespaceTokenizer = WhitespaceTokenizer.INSTANCE;

//Tokenizing the given paragraph
String tokens[] = whitespaceTokenizer.tokenize(sentence);

//Printing the tokens
for(String token : tokens)
    System.out.println(token);
}
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```

javac WhitespaceTokenizerExample.java
java WhitespaceTokenizerExample

```

On executing, the above program reads the given String (raw text), tokenizes it, and displays the following output.

```

Hi.
How
are
you?
Welcome
to
Tutorialspoint.
We
provide
free
tutorials
on
various
technologies

```

## Tokenizing using TokenizerME class

OpenNLP also uses a predefined model, a file named de-token.bin, to tokenize the sentences. It is trained to tokenize the sentences in a given raw text.

The **TokenizerME** class of the **opennlp.tools.tokenizer** package is used to load this model, and tokenize the given raw text using OpenNLP library. To do so, you need to –

- Load the **en-token.bin** model using the **TokenizerModel** class.
- Instantiate the **TokenizerME** class.
- Tokenize the sentences using the **tokenize()** method of this class.

Following are the steps to be followed to write a program which tokenizes the sentences from the given raw text using the **TokenizerME** class.

### Step 1: Loading the model

The model for tokenization is represented by the class named **TokenizerModel**, which belongs to the package **opennlp.tools.tokenize**.

To load a tokenizer model –

- Create an **InputStream** object of the model (Instantiate the **FileInputStream** and pass the path of the model in String format to its constructor).
- Instantiate the **TokenizerModel** class and pass the **InputStream** (object) of the model as a parameter to its constructor, as shown in the following code block.

```
//Loading the Tokenizer model
InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-token.bin");
TokenizerModel tokenModel = new TokenizerModel(inputStream);
```

### Step 2: Instantiating the TokenizerME class

The **TokenizerME** class of the package **opennlp.tools.tokenize** contains methods to chop the raw text into smaller parts (tokens). It uses Maximum Entropy to make its decisions.

Instantiate this class and pass the model object created in the previous step as shown below-

```
//Instantiating the TokenizerME class
TokenizerME tokenizer = new TokenizerME(tokenModel);
```

### Step 3: Tokenizing the sentence

The **tokenize()** method of the **TokenizerME** class is used to tokenize the raw text passed to it. This method accepts a String variable as a parameter, and returns an array of Strings (tokens).

Invoke this method by passing the String format of the sentence to this method, as follows.

```
//Tokenizing the given raw text
String tokens[] = tokenizer.tokenize(paragraph);
```

### Example

Following is the program which tokenizes the given raw text. Save this program in a file with the name **TokenizerMExample.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;

public class TokenizerMExample {

    public static void main(String args[]) throws Exception{

        String sentence = "Hi. How are you? Welcome to Tutorialspoint. "
            + "We provide free tutorials on various technologies";

        //Loading the Tokenizer model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-token.bin");
        TokenizerModel tokenModel = new TokenizerModel(inputStream);

        //Instantiating the TokenizerME class
        TokenizerME tokenizer = new TokenizerME(tokenModel);

        //Tokenizing the given raw text
        String tokens[] = tokenizer.tokenize(sentence);

        //Printing the tokens
        for (String a : tokens)
            System.out.println(a);
    }
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac TokenizerMExample.java
java TokenizerMExample
```

On executing, the above program reads the given String and detects the sentences in it and displays the following output:

```
Hi
.
How
are
you
?
Welcome
to
Tutorialspoint
.
We
provide
free
tutorials
on
various
technologies
```

## Retrieving the Positions of the Tokens

We can also get the positions or **spans** of the tokens using the **tokenizePos()** method. This is the method of the Tokenizer interface of the package **opennlp.tools.tokenize**. Since all the (three) Tokenizer classes implement this interface, you can find this method in all of them.

This method accepts the sentence or raw text in the form of a **string** and returns an array of objects of the type **Span**.

You can get the positions of the tokens using the **tokenizePos()** method, as follows:

```
//Retrieving the tokens
tokenizer.tokenizePos(sentence);
```

## Printing the positions (spans)

The class named **Span** of the **opennlp.tools.util** package is used to store the **start** and **end** integer of sets.

You can store the spans returned by the **tokenizePos()** method in the Span array and print them, as shown in the following code block.

```
//Retrieving the tokens
Span[] tokens = tokenizer.tokenizePos(sentence);
```



```
//Printing the spans of tokens

for( Span token : tokens)
    System.out.println(token);
```

## Printing tokens and their positions together

The **substring()** method of the String class accepts the **begin** and the **end** offsets and returns the respective string. We can use this method to print the tokens and their spans (positions) together, as shown in the following code block.

```
//Printing the spans of tokens
for(Span token : tokens)
    System.out.println(token + " "+sent.substring(token.getStart(), token.getEnd()));
```

## Token and spans Example (SimpleTokenizer)

Following is the program which retrieves the token spans of the raw text using the **SimpleTokenizer** class. It also prints the tokens along with their positions. Save this program in a file with named **SimpleTokenizerSpans.java**.

```
import opennlp.tools.tokenize.SimpleTokenizer;
import opennlp.tools.util.Span;

public class SimpleTokenizerSpans {

    public static void main(String args[]){

        String sent = "Hi. How are you? Welcome to Tutorialspoint. "
            + "We provide free tutorials on various technologies";

        //Instantiating SimpleTokenizer class
        SimpleTokenizer simpleTokenizer = SimpleTokenizer.INSTANCE;

        //Retrieving the boundaries of the tokens
        Span[] tokens =simpleTokenizer.tokenizePos(sent);

        //Printing the spans of tokens
        for( Span token : tokens)
```

```

        System.out.println(token + " " + sent.substring(token.getStart(), token.getEnd()));
    }
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```

javac SimpleTokenizerSpans.java
java SimpleTokenizerSpans

```

On executing, the above program reads the given String (raw text), tokenizes it, and displays the following output:

```

[0..2) Hi
[2..3) .
[4..7) How
[8..11) are
[12..15) you
[15..16) ?
[17..24) Welcome
[25..27) to
[28..42) Tutorialspoint
[42..43) .
[44..46) We
[47..54) provide
[55..59) free
[60..69) tutorials
[70..72) on
[73..80) various
[81..93) technologies

```

### Token and spans Example (WhitespaceTokenizer)

Following is the program which retrieves the token spans of the raw text using the **WhitespaceTokenizer** class. It also prints the tokens along with their positions. Save this program in a file with the name **WhitespaceTokenizerSpans.java**.

```

import opennlp.tools.tokenize.WhitespaceTokenizer;

```

```

import opennlp.tools.util.Span;

public class WhitespaceTokenizerSpans {

    public static void main(String args[]){

        String sent = "Hi. How are you? Welcome to Tutorialspoint. "
            + "We provide free tutorials on various technologies";

        //Instantiating SimpleTokenizer class
        WhitespaceTokenizer whitespaceTokenizer = WhitespaceTokenizer.INSTANCE;

        //Retrieving the tokens
        Span[] tokens =whitespaceTokenizer.tokenizePos(sent);

        //Printing the spans of tokens
        for( Span token : tokens)
            System.out.println(token + " "+sent.substring(token.getStart(), token.getEnd()));
    }
}

```

Compile and execute the saved java file from the command prompt using the following commands

```

javac WhitespaceTokenizerSpans.java
java WhitespaceTokenizerSpans

```

On executing, the above program reads the given String (raw text), tokenizes it, and displays the following output.

```

[0..3) Hi.
[4..7) How
[8..11) are
[12..16) you?
[17..24) Welcome
[25..27) to
[28..43) Tutorialspoint.
[44..46) We
[47..54) provide
[55..59) free

```

```
[60..69) tutorials
[70..72) on
[73..80) various
[81..93) technologies
```

## Token and spans Example (SimpleTokenizer)

Following is the program which retrieves the token spans of the raw text using the **TokenizerME** class. It also prints the tokens along with their positions. Save this program in a file with the name **TokenizerMESpans.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;
import opennlp.tools.util.Span;

public class TokenizerMESpans {
    public static void main(String args[]) throws Exception{
        String sent = "Hello John how are you welcome to Tutorialspoint";

        //Loading the Tokenizer model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-token.bin");
        TokenizerModel tokenModel = new TokenizerModel(inputStream);

        //Instantiating the TokenizerME class
        TokenizerME tokenizer = new TokenizerME(tokenModel);

        //Retrieving the positions of the tokens
        Span tokens[] = tokenizer.tokenizePos(sent);

        //Printing the spans of tokens
        for(Span token : tokens)
            System.out.println(token + " " + sent.substring(token.getStart(), token.getEnd()));
    }
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac TokenizerMESpans.java
java TokenizerMESpans
```

On executing, the above program reads the given String (raw text), tokenizes it, and displays the following output:

```
[0..5) Hello
[6..10) John
[11..14) how
[15..18) are
[19..22) you
[23..30) welcome
[31..33) to
[34..48) Tutorialspoint
```

## Tokenizer Probability

The `getTokenProbabilities()` method of the `TokenizerME` class is used to get the probabilities associated with the most recent calls to the `tokenizePos()` method.

```
//Getting the probabilities of the recent calls to tokenizePos() method
double[] probs = detector.getSentenceProbabilities();
```

Following is the program to print the probabilities associated with the calls to `tokenizePos()` method. Save this program in a file with the name **TokenizerMEProbs.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;
import opennlp.tools.util.Span;

public class TokenizerMEProbs {
    public static void main(String args[]) throws Exception{
        String sent = "Hello John how are you welcome to Tutorialspoint";
        //Loading the Tokenizer model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-token.bin");
        TokenizerModel tokenModel = new TokenizerModel(inputStream);
        //Instantiating the TokenizerME class
        TokenizerME tokenizer = new TokenizerME(tokenModel);
```

```

//Retrieving the positions of the tokens
Span tokens[] = tokenizer.tokenizePos(sent);

//Getting the probabilities of the recent calls to tokenizePos() method
double[] probs = tokenizer.getTokenProbabilities();

//Printing the spans of tokens
for(Span token : tokens)
    System.out.println(token + " "+sent.substring(token.getStart(), token.getEnd()));
System.out.println(" ");
for(int i=0; i<probs.length; i++)
    System.out.println(probs[i]);
}
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```

javac TokenizerMEProbs.java
java TokenizerMEProbs

```

On executing, the above program reads the given String and tokenizes the sentences and prints them. In addition, it also returns the probabilities associated with the most recent calls to the tokenizePos() method.

```

[0..5) Hello
[6..10) John
[11..14) how
[15..18) are
[19..22) you
[23..30) welcome
[31..33) to
[34..48) Tutorialspoint

1.0
1.0
1.0
1.0
1.0

```



1.0  
1.0  
1.0

## 6. Open NLP – Named Entity Recognition

The process of finding names, people, places, and other entities, from a given text is known as **Named Entity Recognition (NER)**. In this chapter, we will discuss how to carry out NER through Java program using OpenNLP library.

### Named Entity Recognition using open NLP

---

To perform various NER tasks, OpenNLP uses different predefined models namely, en-ner-date.bin, en-ner-location.bin, en-ner-organization.bin, en-ner-person.bin, and en-ner-time.bin. All these files are predefined models which are trained to detect the respective entities in a given raw text.

The **opennlp.tools.namefind** package contains the classes and interfaces that are used to perform the NER task. To perform NER task using OpenNLP library, you need to –

- Load the respective model using the **TokenNameFinderModel** class.
- Instantiate the **NameFinder** class.
- Find the names and print them.

Following are the steps to be followed to write a program which detects the name entities from a given raw text.

#### Step 1: Loading the model

The model for sentence detection is represented by the class named **TokenNameFinderModel**, which belongs to the package **opennlp.tools.namefind**.

To load an NER model –

- Create an **InputStream** object of the model (Instantiate the `FileInputStream` and pass the path of the appropriate NER model in String format to its constructor).
- Instantiate the **TokenNameFinderModel** class and pass the **InputStream** (object) of the model as a parameter to its constructor, as shown in the following code block –

```
//Loading the NER-person model
InputStream inputStreamNameFinder = new FileInputStream("C:/OpenNLP_models/en-ner-person.bin");
TokenNameFinderModel model = new TokenNameFinderModel(inputStreamNameFinder);
```

#### Step 2: Instantiating the NameFinderME class

The **NameFinderME** class of the package **opennlp.tools.namefind** contains methods to perform the NER tasks. This class uses the Maximum Entropy model to find the named entities in the given raw text.

Instantiate this class and pass the model object created in the previous step as shown below-

```
//Instantiating the NameFinderME class
NameFinderME nameFinder = new NameFinderME(model);
```

### Step 3: Finding the names in the sentence

The **find()** method of the **NameFinderME** class is used to detect the names in the raw text passed to it. This method accepts a String variable as a parameter.

Invoke this method by passing the String format of the sentence to this method.

```
//Finding the names in the sentence
Span nameSpans[] = nameFinder.find(sentence);
```

### Step 4: Printing the spans of the names in the sentence

The **find()** method of the **NameFinderME** class returns an array of objects of the type **Span**. The class named Span of the **opennlp.tools.util** package is used to store the **start** and **end** integer of sets.

You can store the spans returned by the **find()** method in the Span array and print them, as shown in the following code block.

```
//Printing the sentences and their spans of a sentence
for (Span span : spans)
System.out.println(paragraph.substring(span);
```

## NER Example

Following is the program which reads the given sentence and recognizes the spans of the names of the persons in it. Save this program in a file with the name **NameFinderME\_Example.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;

import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.namefind.TokenNameFinderModel;
import opennlp.tools.util.Span;

public class NameFinderME_Example {
    public static void main(String args[]) throws Exception{
        //Loading the NER - Person model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-ner-person.bin");
        TokenNameFinderModel model = new TokenNameFinderModel(inputStream);
```

```
//Instantiating the NameFinder class
NameFinderME nameFinder = new NameFinderME(model);

//Getting the sentence in the form of String array
String [] sentence = new String[]{
    "Mike",
    "and",
    "Smith",
    "are",
    "good",
    "friends"
};

//Finding the names in the sentence
Span nameSpans[] = nameFinder.find(sentence);

//Printing the spans of the names in the sentence
for(Span s: nameSpans)
    System.out.println(s.toString());
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac NameFinderME_Example.java
java NameFinderME_Example
```

On executing, the above program reads the given String (raw text), detects the names of the persons in it, and displays their positions (spans), as shown below.

```
[0..1) person
[2..3) person
```

## Names along with their Positions

The **substring()** method of the String class accepts the **begin** and the **end offsets** and returns the respective string. We can use this method to print the names and their spans (positions) together, as shown in the following code block.

```
for(Span s: nameSpans)
    System.out.println(s.toString()+" "+tokens[s.getStart()]);
```

Following is the program to detect the names from the given raw text and display them along with their positions. Save this program in a file with the name **NameFinderSentences.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;

import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.namefind.TokenNameFinderModel;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;
import opennlp.tools.util.Span;

public class NameFinderSentences {

    public static void main(String args[]) throws Exception{

        //Loading the tokenizer model
        InputStream inputStreamTokenizer = new FileInputStream("C:/OpenNLP_models/en-token.bin");
        TokenizerModel tokenModel = new TokenizerModel(inputStreamTokenizer);

        //Instantiating the TokenizerME class
        TokenizerME tokenizer = new TokenizerME(tokenModel);

        //Tokenizing the sentence in to a string array
        String sentence = "Mike is senior programming manager and Rama is a clerk both are working at Tutorialspoint";
        String tokens[] = tokenizer.tokenize(sentence);

        //Loading the NER-person model
        InputStream inputStreamNameFinder = new FileInputStream("C:/OpenNLP_models/en-ner-person.bin");
        TokenNameFinderModel model = new TokenNameFinderModel(inputStreamNameFinder);
```

```

//Instantiating the NameFinderME class
NameFinderME nameFinder = new NameFinderME(model);

//Finding the names in the sentence
Span nameSpans[] = nameFinder.find(tokens);

//Printing the names and their spans in a sentence
for(Span s: nameSpans)
    System.out.println(s.toString()+" "+tokens[s.getStart()]);
}
}
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```

javac NameFinderSentences.java
java NameFinderSentences

```

On executing, the above program reads the given String (raw text), detects the names of the persons in it, and displays their positions (spans) as shown below.

```
[0..1) person Mike
```

## Finding the Names of the Location

By loading various models, you can detect various named entities. Following is a Java program which loads the **en-ner-location.bin** model and detects the location names in the given sentence. Save this program in a file with the name **LocationFinder.java**.

```

import java.io.FileInputStream;
import java.io.InputStream;

import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.namefind.TokenNameFinderModel;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;
import opennlp.tools.util.Span;

public class LocationFinder {
    public static void main(String args[]) throws Exception{

```



```

        InputStream inputStreamTokenizer = new FileInputStream("C:/OpenNLP_models/en-
token.bin");
        TokenizerModel tokenModel = new TokenizerModel(inputStreamTokenizer);

        //String paragraph = "Mike and Smith are classmates";
        String paragraph = "Tutorialspoint is located in Hyderabad";

        //Instantiating the TokenizerME class
        TokenizerME tokenizer = new TokenizerME(tokenModel);
        String tokens[] = tokenizer.tokenize(paragraph);

        //Loading the NER-location model
        InputStream inputStreamNameFinder = new FileInputStream("C:/OpenNLP_models/en-
ner-location.bin");
        TokenNameFinderModel model = new TokenNameFinderModel(inputStreamNameFinder);

        //Instantiating the NameFinderME class
        NameFinderME nameFinder = new NameFinderME(model);

        //Finding the names of a location
        Span nameSpans[] = nameFinder.find(tokens);

        //Printing the spans of the locations in the sentence
        for(Span s: nameSpans)
            System.out.println(s.toString()+" "+tokens[s.getStart()]);
    }
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```

javac LocationFinder.java
java LocationFinder

```

On executing, the above program reads the given String (raw text), detects the names of the persons in it, and displays their positions (spans), as shown below.

```
[4..5) location  Hyderabad
```

## NameFinder Probability

The **probs()** method of the **NameFinderME** class is used to get the probabilities of the last decoded sequence.

```
double[] probs = nameFinder.probs();
```

Following is the program to print the probabilities. Save this program in a file with the name **TokenizerMEProbs.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;
import opennlp.tools.util.Span;

public class TokenizerMEProbs {
    public static void main(String args[]) throws Exception{
        String sent = "Hello John how are you welcome to Tutorialspoint";

        //Loading the Tokenizer model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-token.bin");
        TokenizerModel tokenModel = new TokenizerModel(inputStream);

        //Instantiating the TokenizerME class
        TokenizerME tokenizer = new TokenizerME(tokenModel);

        //Retrieving the positions of the tokens
        Span tokens[] = tokenizer.tokenizePos(sent);

        //Getting the probabilities of the recent calls to tokenizePos() method
        double[] probs = tokenizer.getTokenProbabilities();

        //Printing the spans of tokens
        for( Span token : tokens)
            System.out.println(token + " "+sent.substring(token.getStart(), token.getEnd()));
        System.out.println(" ");
        for(int i=0; i<probs.length; i++)
            System.out.println(probs[i]);
    }
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac TokenizerMEProbs.java  
java TokenizerMEProbs
```

On executing, the above program reads the given String, tokenizes the sentences, and prints them. In addition, it also returns the probabilities of the last decoded sequence, as shown below.

```
[0..5) Hello  
[6..10) John  
[11..14) how  
[15..18) are  
[19..22) you  
[23..30) welcome  
[31..33) to  
[34..48) Tutorialspoint  
  
1.0  
1.0  
1.0  
1.0  
1.0  
1.0  
1.0  
1.0  
1.0
```

## 7. OpenNLP – Finding Parts of Speech

Using OpenNLP, you can also detect the Parts of Speech of a given sentence and print them. Instead of full name of the parts of speech, OpenNLP uses short forms of each parts of speech. The following table indicates the various parts of speeches detected by OpenNLP and their meanings.

Part of speech	Meaning of parts of speech
NN	Noun, singular or mass
DT	Determiner
VB	Verb, base form
VBD	Verb, past tense
VBZ	Verb, third person singular present
IN	Preposition or subordinating conjunction
NNP	Proper noun, singular
TO	to
JJ	Adjective

### Tagging the Parts of Speech

To tag the parts of speech of a sentence, OpenNLP uses a model, a file named **en-pos-maxent.bin**. This is a predefined model which is trained to tag the parts of speech of the given raw text.

The **POSTaggerME** class of the **opennlp.tools.postag** package is used to load this model, and tag the parts of speech of the given raw text using OpenNLP library. To do so, you need to –

- Load the **en-pos-maxent.bin** model using the **POSModel** class.
- Instantiate the **POSTaggerME** class.
- Tokenize the sentence.
- Generate the tags using **tag()** method.
- Print the tokens and tags using **POSSample** class.

Following are the steps to be followed to write a program which tags the parts of the speech in the given raw text using the **POSTaggerME** class.

#### Step 1: Load the model

The model for POS tagging is represented by the class named **POSModel**, which belongs to the package **opennlp.tools.postag**.

To load a tokenizer model –

- Create an **InputStream** object of the model (Instantiate the `FileInputStream` and pass the path of the model in String format to its constructor).
- Instantiate the **POSModel** class and pass the **InputStream** (object) of the model as a parameter to its constructor, as shown in the following code block –

```
//Loading Parts of speech-maxent model
InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-pos-maxent.bin");
POSModel model = new POSModel(inputStream);
```

## Step 2: Instantiating the POSTaggerME class

The **POSTaggerME** class of the package **opennlp.tools.postag** is used to predict the parts of speech of the given raw text. It uses Maximum Entropy to make its decisions.

Instantiate this class and pass the model object created in the previous step, as shown below-

```
//Instantiating POSTaggerME class
POSTaggerME tagger = new POSTaggerME(model);
```

## Step 3: Tokenizing the sentence

The **tokenize()** method of the **whitespaceTokenizer** class is used to tokenize the raw text passed to it. This method accepts a String variable as a parameter, and returns an array of Strings (tokens).

Instantiate the **whitespaceTokenizer** class and then invoke this method by passing the String format of the sentence to this method.

```
//Tokenizing the sentence using WhitespaceTokenizer class
WhitespaceTokenizer whitespaceTokenizer= WhitespaceTokenizer.INSTANCE;
String[] tokens = whitespaceTokenizer.tokenize(sentence);
```

## Step 4: Generating the tags

The **tag()** method of the **whitespaceTokenizer** class assigns POS tags to the sentence of tokens. This method accepts an array of tokens (String) as a parameter and returns tag (array).

Invoke the **tag()** method by passing the tokens generated in the previous step to it.

```
//Generating tags
String[] tags = tagger.tag(tokens);
```

## Step 5: Printing the tokens and the tags

The **POSSample** class represents the POS-tagged sentence. To instantiate this class, we would require an array of tokens (of the text) and an array of tags.

The **toString()** method of this class returns the tagged sentence. Instantiate this class by passing the token and the tag arrays created in the previous steps and invoke its **toString()** method, as shown in the following code block.

```
//Instantiating the POSSample class
POSSample sample = new POSSample(tokens, tags);
System.out.println(sample.toString());
```

## Example

Following is the program which tags the parts of speech in a given raw text. Save this program in a file with the name **PosTaggerExample.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;

import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSSample;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.tokenize.WhitespaceTokenizer;

public class PosTaggerExample {

    public static void main(String args[]) throws Exception{

        //Loading Parts of speech-maxent model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-pos-maxent.bin");
        POSModel model = new POSModel(inputStream);

        //Instantiating POSTaggerME class
        POSTaggerME tagger = new POSTaggerME(model);

        String sentence = "Hi welcome to Tutorialspoint";

        //Tokenizing the sentence using WhitespaceTokenizer class
        WhitespaceTokenizer whitespaceTokenizer= WhitespaceTokenizer.INSTANCE;
        String[] tokens = whitespaceTokenizer.tokenize(sentence);

        //Generating tags
        String[] tags = tagger.tag(tokens);
```

```
//Instantiating the POSSample class
POSSample sample = new POSSample(tokens, tags);

System.out.println(sample.toString());

}

}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac PosTaggerExample.java
java PosTaggerExample
```

On executing, the above program reads the given text and detects the parts of speech of these sentences and displays them, as shown below.

```
Hi_NNP welcome_JJ to_TO Tutorialspoint_VB
```

## POS Tagger Performance

Following is the program which tags the parts of speech of a given raw text. It also monitors the performance and displays the performance of the tagger. Save this program in a file with the name **PosTagger\_Performance.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;

import opennlp.tools.cmdline.PerformanceMonitor;
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSSample;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.tokenize.WhitespaceTokenizer;

public class PosTagger_Performance {
    public static void main(String args[]) throws Exception{
        //Loading Parts of speech-maxent model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-pos-maxent.bin");
        POSModel model = new POSModel(inputStream);

        //Creating an object of WhitespaceTokenizer class
        WhitespaceTokenizer whitespaceTokenizer= WhitespaceTokenizer.INSTANCE;
```



```

//Tokenizing the sentence
String sentence = "Hi welcome to Tutorialspoint";

String[] tokens = whitespaceTokenizer.tokenize(sentence);

//Instantiating POSTaggerME class
POSTaggerME tagger = new POSTaggerME(model);

//Generating tags
String[] tags = tagger.tag(tokens);

//Instantiating POSSample class
POSSample sample = new POSSample(tokens, tags);
System.out.println(sample.toString());

//Monitoring the performance of POS tagger
PerformanceMonitor perfMon = new PerformanceMonitor(System.err, "sent");
perfMon.start();
perfMon.incrementCounter();
perfMon.stopAndPrintFinalResult();
}
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```

javac PosTaggerExample.java
java PosTaggerExample

```

On executing, the above program reads the given text and tags the parts of speech of these sentences and displays them. In addition, it also monitors the performance of the POS tagger and displays it.

```

Hi_NNP welcome_JJ to_TO Tutorialspoint_VB

Average: 0.0 sent/s
Total: 1 sent
Runtime: 0.0s

```

## POS Tagger Probability

The **probs()** method of the **POSTaggerME** class is used to find the probabilities for each tag of the recently tagged sentence.

```
//Getting the probabilities of the recent calls to tokenizePos() method
double[] probs = detector.getSentenceProbabilities();
```

Following is the program which displays the probabilities for each tag of the last tagged sentence. Save this program in a file with the name **PosTaggerProbs.java**

```
import java.io.FileInputStream;
import java.io.InputStream;

import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSSample;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.tokenize.WhitespaceTokenizer;

public class PosTaggerProbs {
    public static void main(String args[]) throws Exception{
        //Loading Parts of speech-maxent model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-pos-maxent.bin");
        POSModel model = new POSModel(inputStream);

        //Creating an object of WhitespaceTokenizer class
        WhitespaceTokenizer whitespaceTokenizer= WhitespaceTokenizer.INSTANCE;

        //Tokenizing the sentence
        String sentence = "Hi welcome to Tutorialspoint";
        String[] tokens = whitespaceTokenizer.tokenize(sentence);

        //Instantiating POSTaggerME class
        POSTaggerME tagger = new POSTaggerME(model);

        //Generating tags
        String[] tags = tagger.tag(tokens);
        //Instantiating the POSSample class
        POSSample sample = new POSSample(tokens, tags);

        System.out.println(sample.toString());
    }
}
```

```
//Probabilities for each tag of the last tagged sentence.  
double [] probs = tagger.probs();  
System.out.println(" ");  
//Printing the probabilities  
for(int i=0; i<probs.length; i++)  
    System.out.println(probs[i]);  
}  
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac TokenizerMEProbs.java  
java TokenizerMEProbs
```

On executing, the above program reads the given raw text, tags the parts of speech of each token in it, and displays them. In addition, it also displays the probabilities for each parts of speech in the given sentence, as shown below.

```
Hi_NNP welcome_JJ to_TO Tutorialspoint_VB  
  
0.6416834779738033  
0.42983612874819177  
0.8584513635863117  
0.4394784478206072
```

## 8. OpenNLP – Parsing the Sentences

Using OpenNLP API, you can parse the given sentences. In this chapter, we will discuss how to parse raw text using OpenNLP API.

### Parsing Raw Text using OpenNLP Library

---

To detect the sentences, OpenNLP uses a predefined model, a file named **en-parser-chunking.bin**. This is a predefined model which is trained to parse the given raw text.

The **Parser** class of the **opennlp.tools.Parser** package is used to hold the parse constituents and the **ParserTool** class of the **opennlp.tools.cmdline.parser** package is used to parse the content.

Following are the steps to be followed to write a program which parses the given raw text using the **ParserTool** class.

#### Step 1: Loading the model

The model for parsing text is represented by the class named **ParserModel**, which belongs to the package **opennlp.tools.parser**.

To load a tokenizer model –

- Create an **InputStream** object of the model (Instantiate the `FileInputStream` and pass the path of the model in String format to its constructor).
- Instantiate the **ParserModel** class and pass the **InputStream** (object) of the model as a parameter to its constructor, as shown in the following code block –

```
//Loading parser model  
  
InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-parser-  
chunking.bin");  
  
ParserModel model = new ParserModel(inputStream);
```

#### Step 2: Creating an object of the Parser class

The **Parser** class of the package **opennlp.tools.parser** represents a data structure for holding parse constituents. You can create an object of this class using the static **create()** method of the **ParserFactory** class.

Invoke the **create()** method of the **ParserFactory** by passing the model object created in the previous step, as shown below:

```
//Creating a parser  
  
Parser parser = ParserFactory.create(model);
```

### Step 3: Parsing the sentence

The **parseLine()** method of the **ParserTool** class is used to parse the raw text in OpenNLP. This method accepts –

- a String variable representing the text to be parsed.
- a parser object.
- an integer representing the number of parses to be carried out.

Invoke this method by passing the sentence the following parameters: the parse object created in the previous steps, and an integer representing the required number of parses to be carried out.

```
//Parsing the sentence
String sentence = "Tutorialspoint is the largest tutorial library.";
Parse topParses[] = ParserTool.parseLine(sentence, parser, 1);
```

### Example

Following is the program which parses the given raw text. Save this program in a file with the name **ParserExample.java**.

```
import java.io.FileInputStream;
import java.io.InputStream;

import opennlp.tools.cmdline.parser.ParserTool;
import opennlp.tools.parser.Parse;
import opennlp.tools.parser.Parser;
import opennlp.tools.parser.ParserFactory;
import opennlp.tools.parser.ParserModel;

public class ParserExample {
    public static void main(String args[]) throws Exception{

        //Loading parser model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-parser-chunking.bin");
        ParserModel model = new ParserModel(inputStream);

        //Creating a parser
        Parser parser = ParserFactory.create(model);

        //Parsing the sentence
        String sentence = "Tutorialspoint is the largest tutorial library.";
```

```
Parse topParses[] = ParserTool.parseLine(sentence, parser, 1);

for (Parse p : topParses)
    p.show();
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac ParserExample.java
java ParserExample
```

On executing, the above program reads the given raw text, parses it, and displays the following output:

```
(TOP (S (NP (NN Tutorialspoint)) (VP (VBZ is) (NP (DT the) (JJ largest) (NN
tutorial) (NN library.)))))
```

## 9. OpenNLP – Chunking Sentences

Chunking a sentences refers to breaking/dividing a sentence into parts of words such as word groups and verb groups

### Chunking a Sentence using OpenNLP

---

To detect the sentences, OpenNLP uses a model, a file named **en-chunker.bin**. This is a predefined model which is trained to chunk the sentences in the given raw text.

The **opennlp.tools.chunker** package contains the classes and interfaces that are used to find non-recursive syntactic annotation such as noun phrase chunks.

You can chunk a sentence using the method **chunk()** of the **ChunkerME** class. This method accepts tokens of a sentence and POS tags as parameters. Therefore, before starting the process of chunking, first of all you need to Tokenize the sentence and generate the parts POS tags of it.

To chunk a sentence using OpenNLP library, you need to –

- Tokenize the sentence.
- Generate POS tags for it.
- Load the **en-chunker.bin** model using the **ChunkerModel** class
- Instantiate the **ChunkerME** class
- Chunk the sentences using the **chunk()** method of this class.

Following are the steps to be followed to write a program to chunk sentences from the given raw text.

#### Step 1: Tokenizing the sentence

Tokenize the sentences using the **tokenize()** method of the **WhitespaceTokenizer** class, as shown in the following code block.

```
//Tokenizing the sentence
String sentence = "Hi welcome to Tutorialspoint";
WhitespaceTokenizer whitespaceTokenizer= WhitespaceTokenizer.INSTANCE;
String[] tokens = whitespaceTokenizer.tokenize(sentence);
```

#### Step 2: Generating the POS tags

Generate the POS tags of the sentence using the **tag()** method of the **POSTaggerME** class, as shown in the following code block.

```
//Generating the POS tags
File file = new File("C:/OpenNLP_models/en-pos-maxent.bin");
POSModel model = new POSModelLoader().load(file);
//Constructing the tagger
POSTaggerME tagger = new POSTaggerME(model);
//Generating tags from the tokens
String[] tags = tagger.tag(tokens);
```

### Step 3: Loading the model

The model for chunking a sentence is represented by the class named **ChunkerModel**, which belongs to the package **opennlp.tools.chunker**.

To load a sentence detection model –

- Create an **InputStream** object of the model (Instantiate the **FileInputStream** and pass the path of the model in String format to its constructor).
- Instantiate the **ChunkerModel** class and pass the **InputStream** (object) of the model as a parameter to its constructor, as shown in the following code block –

```
//Loading the chunker model
InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-chunker.bin");
ChunkerModel chunkerModel = new ChunkerModel(inputStream);
```

### Step 4: Instantiating the chunkerME class

The **chunkerME** class of the package **opennlp.tools.chunker** contains methods to chunk the sentences. This is a maximum-entropy-based chunker.

Instantiate this class and pass the model object created in the previous step.

```
//Instantiate the ChunkerME class
ChunkerME chunkerME = new ChunkerME(chunkerModel);
```

### Step 5: Chunking the sentence

The **chunk()** method of the **ChunkerME** class is used to chunk the sentences in the raw text passed to it. This method accepts two String arrays representing tokens and tags, as parameters.

Invoke this method by passing the token array and tag array created in the previous steps as parameters.

```
//Generating the chunks
String result[] = chunkerME.chunk(tokens, tags);
```



## Example

Following is the program to chunk the sentences in the given raw text. Save this program in a file with the name **ChunkerExample.java**.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

import opennlp.tools.chunker.ChunkerME;
import opennlp.tools.chunker.ChunkerModel;
import opennlp.tools.cmdline.postag.POSModelLoader;
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.tokenize.WhitespaceTokenizer;

public class ChunkerExample{
    public static void main(String args[]) throws IOException {
        //Tokenizing the sentence
        String sentence = "Hi welcome to Tutorialspoint";
        WhitespaceTokenizer whitespaceTokenizer= WhitespaceTokenizer.INSTANCE;
        String[] tokens = whitespaceTokenizer.tokenize(sentence);

        //Generating the POS tags
        //Load the parts of speech model
        File file = new File("C:/OpenNLP_models/en-pos-maxent.bin");
        POSModel model = new POSModelLoader().load(file);
        //Constructing the tagger
        POSTaggerME tagger = new POSTaggerME(model);
        //Generating tags from the tokens
        String[] tags = tagger.tag(tokens);

        //Loading the chunker model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-chunker.bin");
        ChunkerModel chunkerModel = new ChunkerModel(inputStream);

        //Instantiate the ChunkerME class
        ChunkerME chunkerME = new ChunkerME(chunkerModel);
```

```
//Generating the chunks

String result[] = chunkerME.chunk(tokens, tags);

for (String s : result)
    System.out.println(s);
}
}
```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac ChunkerExample.java
java ChunkerExample
```

On executing, the above program reads the given String and chunks the sentences in it, and displays them as shown below.

```
Loading POS Tagger model ... done (1.040s)
B-NP
I-NP
B-VP
I-VP
```

## Detecting the Positions of the Tokens

We can also detect the positions or spans of the chunks using the **chunkAsSpans()** method of the **ChunkerME** class. This method returns an array of objects of the type **Span**. The class named Span of the **opennlp.tools.util** package is used to store the **start** and **end** integer of sets.

You can store the spans returned by the **chunkAsSpans()** method in the Span array and print them, as shown in the following code block.

```
//Generating the tagged chunk spans
Span[] span = chunkerME.chunkAsSpans(tokens, tags);

for (Span s : span)
    System.out.println(s.toString());
```

## Example

Following is the program which detects the sentences in the given raw text. Save this program in a file with the name **ChunkerSpansEample.java**.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

import opennlp.tools.chunker.ChunkerME;
import opennlp.tools.chunker.ChunkerModel;
import opennlp.tools.cmdline.postag.POSModelLoader;
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.tokenize.WhitespaceTokenizer;
import opennlp.tools.util.Span;

public class ChunkerSpansEample{
    public static void main(String args[]) throws IOException {
        //Load the parts of speech model
        File file = new File("C:/OpenNLP_models/en-pos-maxent.bin");
        POSModel model = new POSModelLoader().load(file);

        //Constructing the tagger
        POSTaggerME tagger = new POSTaggerME(model);

        //Tokenizing the sentence
        String sentence = "Hi welcome to Tutorialspoint";
        WhitespaceTokenizer whitespaceTokenizer= WhitespaceTokenizer.INSTANCE;
        String[] tokens = whitespaceTokenizer.tokenize(sentence);

        //Generating tags from the tokens
        String[] tags = tagger.tag(tokens);

        //Loading the chunker model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-chunker.bin");
        ChunkerModel chunkerModel = new ChunkerModel(inputStream);
```

```

    ChunkerME chunkerME = new ChunkerME(chunkerModel);

    //Generating the tagged chunk spans
    Span[] span = chunkerME.chunkAsSpans(tokens, tags);

    for (Span s : span)
        System.out.println(s.toString());
    }
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```

javac ChunkerSpansEample.java
java ChunkerSpansEample

```

On executing, the above program reads the given String and spans of the chunks in it, and displays the following output:

```

Loading POS Tagger model ... done (1.059s)
[0..2) NP
[2..4) VP

```

## Chunker Probability Detection

The **probs()** method of the **ChunkerME** class returns the probabilities of the last decoded sequence.

```

//Getting the probabilities of the last decoded sequence
double[] probs = chunkerME.probs();

```

Following is the program to print the probabilities of the last decoded sequence by the **chunker**. Save this program in a file with the name **ChunkerProbsExample.java**.

```

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import opennlp.tools.chunker.ChunkerME;
import opennlp.tools.chunker.ChunkerModel;
import opennlp.tools.cmdline.postag.POSModelLoader;
import opennlp.tools.postag.POSModel;

```

```

import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.tokenize.WhitespaceTokenizer;

public class ChunkerProbsExample{
    public static void main(String args[]) throws IOException {
        //Load the parts of speech model
        File file = new File("C:/OpenNLP_models/en-pos-maxent.bin");
        POSModel model = new POSModelLoader().load(file);

        //Constructing the tagger
        POSTaggerME tagger = new POSTaggerME(model);

        //Tokenizing the sentence
        String sentence = "Hi welcome to Tutorialspoint";
        WhitespaceTokenizer whitespaceTokenizer= WhitespaceTokenizer.INSTANCE;
        String[] tokens = whitespaceTokenizer.tokenize(sentence);

        //Generating tags from the tokens
        String[] tags = tagger.tag(tokens);

        //Loading the chunker model
        InputStream inputStream = new FileInputStream("C:/OpenNLP_models/en-chunker.bin");
        ChunkerModel cModel = new ChunkerModel(inputStream);
        ChunkerME chunkerME = new ChunkerME(cModel);

        //Generating the chunk tags
        chunkerME.chunk(tokens, tags);

        //Getting the probabilities of the last decoded sequence
        double[] probs = chunkerME.probs();
        for(int i=0; i<probs.length; i++)
            System.out.println(probs[i]);
    }
}

```

Compile and execute the saved Java file from the Command prompt using the following commands:

```
javac ChunkerProbsExample.java  
java ChunkerProbsExample
```

On executing, the above program reads the given String, chunks it, and prints the probabilities of the last decoded sequence.

```
0.9592746040797778  
0.6883933131241501  
0.8830563473996004  
0.8951150529746051
```

# 10. OpenNLP – Command Line Interface

OpenNLP provides a Command Line Interface (CLI) to carry out different operations through the command line. In this chapter, we will take some examples to show how we can use the OpenNLP Command Line Interface.

## Tokenizing

---

### input.txt

```
Hi. How are you? Welcome to Tutorialspoint. We provide free tutorials on various technologies
```

### Syntax

```
> opennlp TokenizerME path_for_models../en-token.bin <inputfile..> outputfile..
```

### command

```
C:\EXAMPLES\opennlp> opennlp TokenizerME C:\OpenNLP_models/en-token.bin <input.txt  
>output.txt
```

### output

```
Loading Tokenizer model ... done (0.207s)
```

```
Average: 214.3 sent/s
```

```
Total: 3 sent
```

```
Runtime: 0.014s
```

### output.txt

```
Hi . How are you ? Welcome to Tutorialspoint . We provide free tutorials on various technologies
```

## Sentence Detection

---

### input.txt

```
Hi. How are you? Welcome to Tutorialspoint. We provide free tutorials on various technologies
```

### Syntax

```
> opennlp SentenceDetector path_for_models../en-token.bin <inputfile..> outputfile..
```

### command

```
C:\EXAMPLES\opennlp> opennlp SentenceDetector C:\OpenNLP_models/en-sent.bin  
<input.txt > output_sendet.txt
```

### Output

```
Loading Sentence Detector model ... done (0.067s)
```

```
Average: 750.0 sent/s
```

```
Total: 3 sent
```

```
Runtime: 0.004s
```

### Output\_sendet.txt

```
Hi. How are you?  
Welcome to Tutorialspoint.  
We provide free tutorials on various technologies
```

## Named Entity Recognition

---

### input.txt

```
<START:person> <START:person> Mike <END> <END> is senior programming manager and  
<START:person> Rama <END> is a clerk both are working at Tutorialspoint
```

### Syntax

```
> opennlp TokenNameFinder path_for_models../en-token.bin <inputfile..>
```



## Command

```
C:\EXAMPLES\opennlp>opennlp TokenNameFinder C:\OpenNLP_models\en-ner-person.bin


```

## Output

```
Loading Token Name Finder model ... done (0.730s)
<START:person> <START:person> Mike <END> <END> is senior programming manager and
<START:person> Rama <END> is a clerk both are working at Tutorialspoint

Average: 55.6 sent/s
Total: 1 sent
Runtime: 0.018s
```

## Parts of Speech Tagging

### Input.txt

```
Hi. How are you? Welcome to Tutorialspoint. We provide free tutorials on various
technologies
```

### Syntax

```
> opennlp POSTagger path_for_models../en-token.bin <inputfile..
```

## Command

```
C:\EXAMPLES\opennlp>opennlp POSTagger C:\OpenNLP_models/en-pos-maxent.bin < input.txt
```

## Output

```
Loading POS Tagger model ... done (1.315s)
Hi._NNP How_WRB are_VBP you?_JJ Welcome_NNP to_TO Tutorialspoint._NNP We_PRP
provide_VBP free_JJ tutorials_NNS on_IN various_JJ technologies_NNS

Average: 66.7 sent/s
Total: 1 sent
Runtime: 0.015s
```