

TIKA - QUICK GUIDE

http://www.tutorialspoint.com/tika/tika_quick_guide.htm

Copyright © tutorialspoint.com

TIKA - OVERVIEW

What is Apache Tika?

- Apache Tika is a library that is used for document type detection and content extraction from various file formats.
- Internally, Tika uses existing various document parsers and document type detection techniques to detect and extract data.
- Using Tika, one can develop a universal type detector and content extractor to extract both structured text as well as metadata from different types of documents such as spreadsheets, text documents, images, PDFs and even multimedia input formats to a certain extent.
- Tika provides a single generic API for parsing different file formats. It uses 83 existing specialized parser libraries for each document type.
- All these parser libraries are encapsulated under a single interface called the **Parser interface**.



Why Tika?

According to filext.com, there are about 15k to 51k content types, and this number is growing day by day. Data is being stored in various formats such as text documents, excel spreadsheet, PDFs, images, and multimedia files, to name a few. Therefore, applications such as search engines and content management systems need additional support for easy extraction of data from these document types. Apache Tika serves this purpose by providing a generic API to locate and extract data from multiple file formats.

Apache Tika Applications

There are various applications that make use of Apache Tika. Here we will discuss a few prominent applications that depend heavily on Apache Tika.

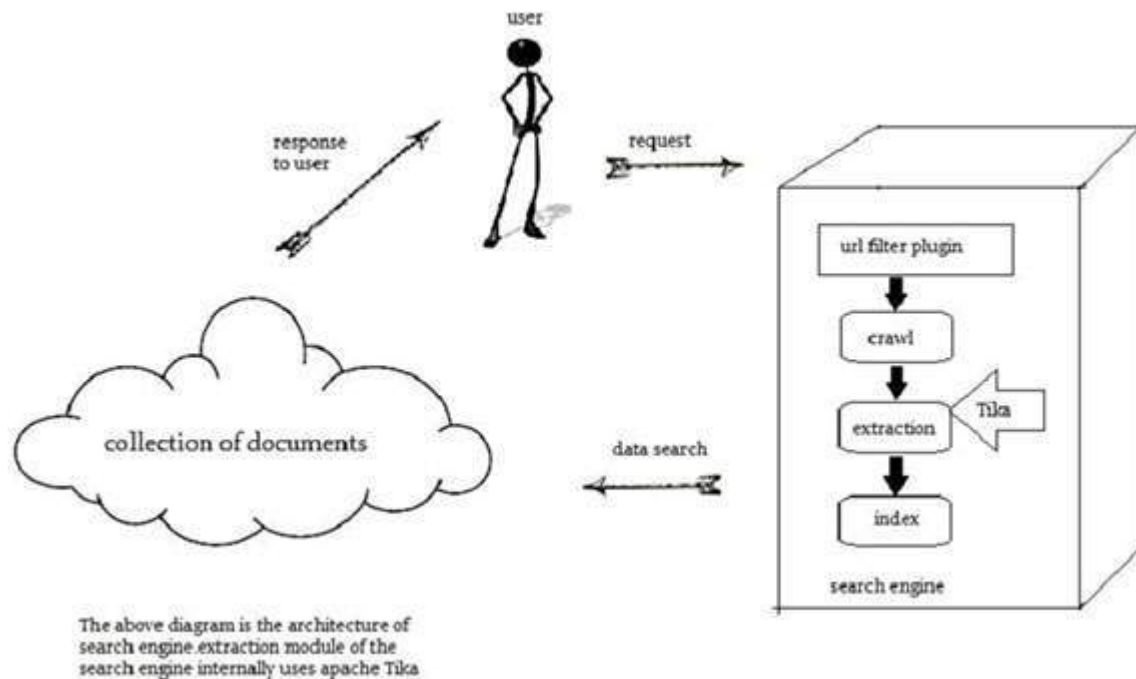
Search Engines

Tika is widely used while developing search engines to index the text contents of digital documents.

- Search engines are information processing systems designed to search information and indexed documents from the Web.
- Crawler is an important component of a search engine that crawls through the Web to fetch

the documents that are to be indexed using some indexing technique. Thereafter, the crawler transfers these indexed documents to an extraction component.

- The duty of extraction component is to extract the text and metadata from the document. Such extracted content and metadata are very useful for a search engine. This extraction component contains Tika.
- The extracted content is then passed to the indexer of the search engine that uses it to build a search index. Apart from this, the search engine uses the extracted content in many other ways as well.



Document Analysis

- In the field of artificial intelligence, there are certain tools to analyze documents automatically at semantic level and extract all kinds of data from them.
- In such applications, the documents are classified based on the prominent terms in the extracted content of the document.
- These tools make use of Tika for content extraction to analyze documents varying from plain text to digital documents.

Digital Asset Management

- Some organizations manage their digital assets such as photographs, e-books, drawings, music and video using a special application known as digital asset management *DAM*.
- Such applications take the help of document type detectors and metadata extractor to classify the various documents.

Content Analysis

- Websites like Amazon recommend newly released contents of their website to individual users according to their interests. To do so, these websites follow **machine learning techniques**, or take the help of social media websites like Facebook to extract required information such as likes and interests of the users. This gathered information will be in the form of html tags or other formats that require further content type detection and extraction.
- For content analysis of a document, we have technologies that implement machine learning techniques such as **UIMA** and **Mahout**. These technologies are useful in clustering and

analyzing the data in the documents.

- **Apache Mahout** is a framework which provides ML algorithms on Apache Hadoop – a cloud computing platform. Mahout provides an architecture by following certain clustering and filtering techniques. By following this architecture, programmers can write their own ML algorithms to produce recommendations by taking various text and metadata combinations. To provide inputs to these algorithms, recent versions of Mahout use Tika to extract text and metadata from binary content.
- **Apache UIMA** analyzes and processes various programming languages and produces UIMA annotations. Internally it uses Tika Annotator to extract document text and metadata.

History

Year	Development
2006	The idea of Tika was projected before the Lucene Project Management Committee.
2006	The concept of Tika and its usefulness in the Jackrabbit project was discussed.
2007	Tika entered into Apache incubator.
2008	Versions 0.1 and 0.2 were released and Tika graduated from the incubator to the Lucene sub-project.
2009	Versions 0.3, 0.4, and 0.5 were released.
2010	Version 0.6 and 0.7 were released and Tika graduated into the top-level Apache project.
2011	Tika 1.0 was released and the book on Tika "Tika in Action" was also released in the same year.

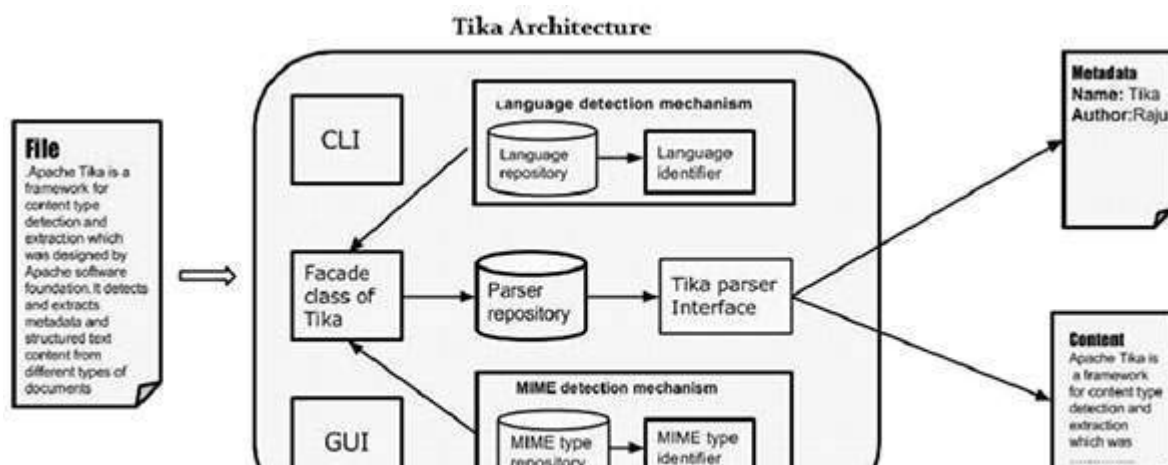
TIKA - ARCHITECTURE

Application-Level Architecture of Tika

Application programmers can easily integrate Tika in their applications. Tika provides a Command Line Interface and a GUI to make it user friendly.

In this chapter, we will discuss the four important modules that constitute the Tika architecture. The following illustration shows the architecture of Tika along with its four modules:

- Language detection mechanism.
- MIME detection mechanism.
- Parser interface.
- Tika Facade class.



Language Detection Mechanism

Whenever a text document is passed to Tika, it will detect the language in which it was written. It accepts documents without language annotation and adds that information in the metadata of the document by detecting the language.

To support language identification, Tika has a class called **Language Identifier** in the package **org.apache.tika.language**, and a language identification repository inside which contains algorithms for language detection from a given text. Tika internally uses N-gram algorithm for language detection.

MIME Detection Mechanism

Tika can detect the document type according to the MIME standards. Default MIME type detection in Tika is done using [org.apache.tika.mime.MimeTypes](#). It uses the [org.apache.tika.detect.Detector](#) interface for most of the content type detection.

Internally Tika uses several techniques like file globs, content-type hints, magic bytes, character encodings, and several other techniques.

Parser Interface

The parser interface of `org.apache.tika.parser` is the key interface for parsing documents in Tika. This Interface extracts the text and the metadata from a document and summarizes it for external users who are willing to write parser plugins.

Using different concrete parser classes, specific for individual document types, Tika supports a lot of document formats. These format specific classes provide support for different document formats, either by directly implementing the parser logic or by using external parser libraries.

Tika Facade Class

Using Tika facade class is the simplest and direct way of calling Tika from Java, and it follows the facade design pattern. You can find the Tika facade class in the `org.apache.tika` package of Tika API.

By implementing basic use cases, Tika acts as a broker of landscape. It abstracts the underlying complexity of the Tika library such as MIME detection mechanism, parser interface, and language detection mechanism, and provides the users a simple interface to use.

Features of Tika

- **Unified parser Interface** : Tika encapsulates all the third party parser libraries within a single parser interface. Due to this feature, the user escapes from the burden of selecting the suitable parser library and use it according to the file type encountered.
- **Low memory usage** : Tika consumes less memory resources therefore it is easily embeddable with Java applications. We can also use Tika within the application which run on platforms with less resources like mobile PDA.
- **Fast processing** : Quick content detection and extraction from applications can be expected.
- **Flexible metadata** : Tika understands all the metadata models which are used to describe files.
- **Parser integration** : Tika can use various parser libraries available for each document type in a single application.
- **MIME type detection** : Tika can detect and extract content from all the media types included in the MIME standards.

- **Language detection** : Tika includes language identification feature, therefore can be used in documents based on language type in a multi lingual websites.

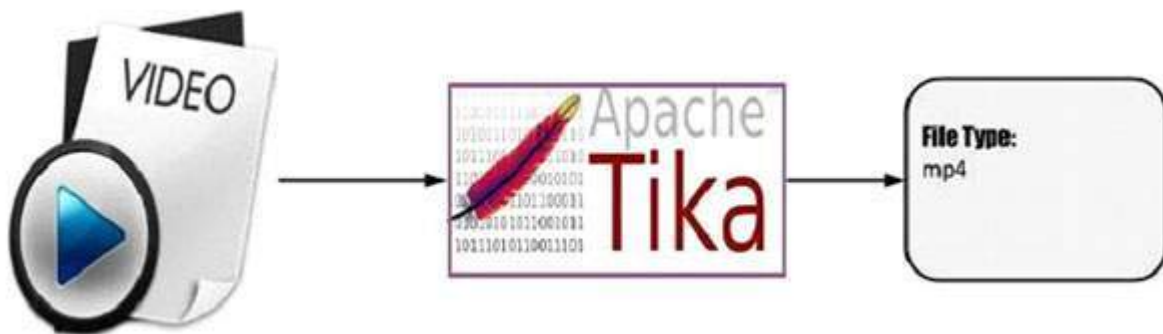
Functionalities of Tika

Tika supports various functionalities:

- Document type detection
- Content extraction
- Metadata extraction
- Language detection

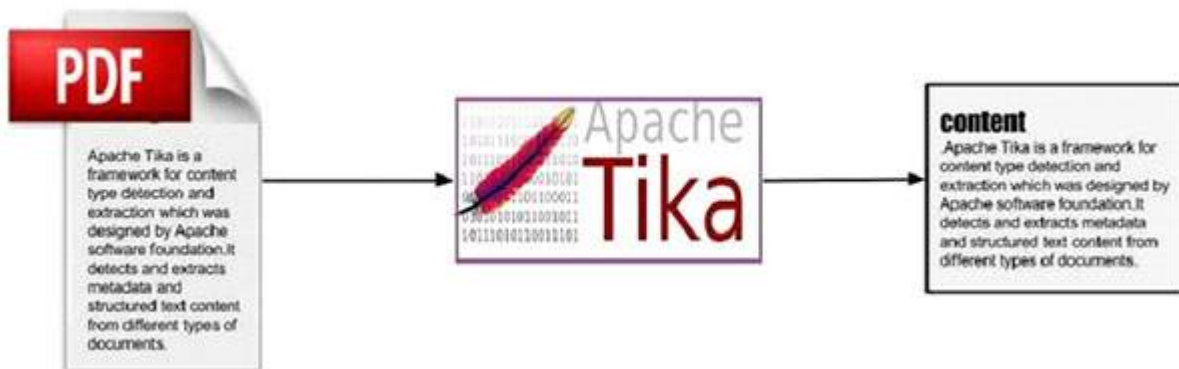
Document Type Detection

Tika uses various detection techniques and detects the type of the document given to it.



Content Extraction

Tika has a parser library that can parse the content of various document formats and extract them. After detecting the type of the document, it selects the appropriate parser from the parser repository and passes the document. Different classes of Tika have methods to parse different document formats.



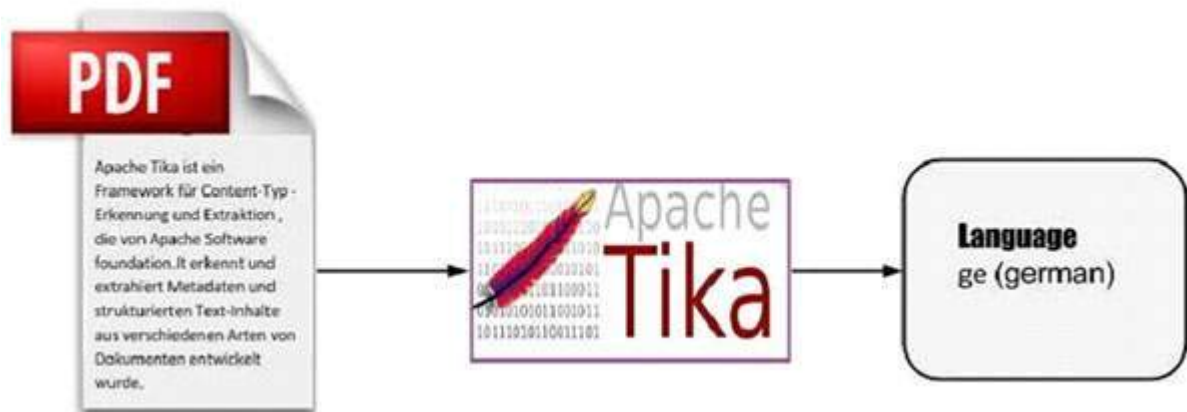
Metadata Extraction

Along with the content, Tika extracts the metadata of the document with the same procedure as in content extraction. For some document types, Tika have classes to extract metadata.



Language Detection

Internally, Tika follows algorithms like n-gram to detect the language of the content in a given document. Tika depends on classes like Language identifier and Profiler for language identification.



TIKA - ENVIRONMENT

This chapter takes you through the process of setting up Apache Tika on Windows and Linux. User administration is needed while installing the Apache Tika.

System Requirements

JDK	Java SE 2 JDK 1.6 or above
Memory	1 GB RAM <i>recommended</i>
Disk Space	No minimum requirement
Operating System Version	Windows XP or above, Linux

Step 1: Verifying Java Installation

To verify Java installation, open the console and execute the following java command.

OS	Task	Command
Windows	Open command console	\>java -version
Linux	Open command terminal	\$java -version

If Java has been installed properly on your system, then you should get one of the following outputs, depending on the platform you are working on.

OS	Output
Windows	Java version "1.7.0_60" Java <i>TM</i> SE Run Time Environment <i>build</i> 1.7.0.60 – b19 Java Hotspot <i>TM</i> 64-bit Server VM <i>build</i> 24.60 – b09, <i>mixedmode</i>

Linux

java version "1.7.0_25"

Open JDK Runtime Environment *rhel – 2.3.10.4.el6₄ – x86₆₄*

Open JDK 64-Bit Server VM *build23.7 – b01, mixedmode*

- We assume the readers of this tutorial have Java 1.7.0_60 installed on their system before proceeding for this tutorial.
- In case you do not have Java SDK, download its current version from <http://www.oracle.com/technetwork/java/javase/downloads/index.html> and have it installed.

Step 2: Setting Java Environment

Set the JAVA_HOME environment variable to point to the base directory location where Java is installed on your machine. For example,

OS	Output
Windows	Set Environmental variable JAVA_HOME to C:\ProgramFiles\java\jdk1.7.0_60
Linux	export JAVA_HOME=/usr/local/java-current

Append the full path of the Java compiler location to the System Path.

OS	Output
Windows	Append the String; C:\Program Files\Java\jdk1.7.0_60\bin to the end of the system variable PATH.
Linux	export PATH=PATH:JAVA_HOME/bin/

Verify the command java-version from command prompt as explained above.

Step 3: Setting up Apache Tika Environment

Programmers can integrate Apache Tika in their environment by using

- Command line,
- Tika API,
- Command line interface *CLI* of Tika,
- Graphical User interface *GUI* of Tika, or
- the source code.

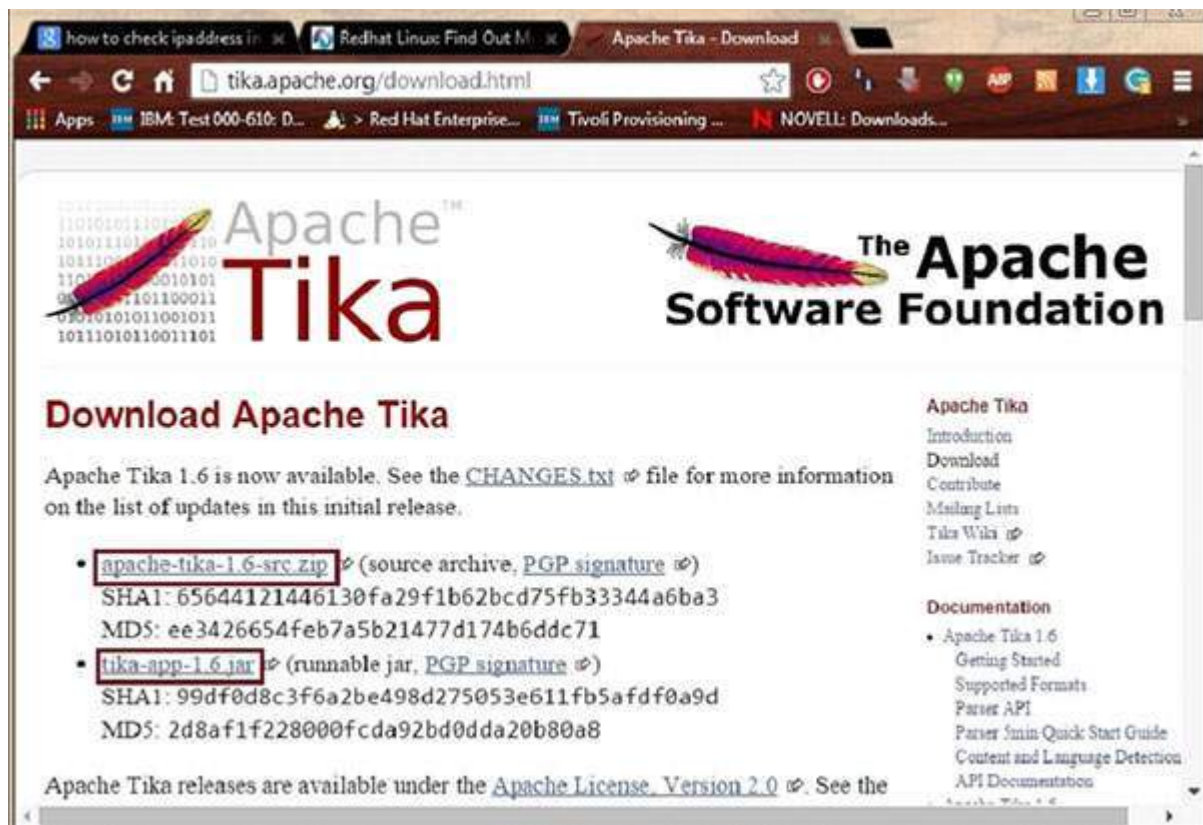
For any of these approaches, first of all, you have to download the source code of Tika.

You will find the source code of Tika at <http://Tika.apache.org/download.html>, where you will find two links:

apache-tika-1.6-src.zip: It contains the source code of Tika, and **Tika -app-1.6.jar**: It is a jar file that contains the Tika application.

Download these two files. A snapshot of the official website of Tika is shown below.





After downloading the files, set the classpath for the jar file **tika-app-1.6.jar**. Add the complete path of the jar file as shown in the table below.

OS	Output
Windows	Append the String "C:\jars\Tika-app-1.6.jar" to the user environment variable CLASSPATH
Linux	Export CLASSPATH=\$CLASSPATH: /usr/share/jars/Tika-app-1.6.jar:

Apache provides Tika application, a Graphical User Interface *GUI* application using Eclipse.

Tika-Maven Build using Eclipse

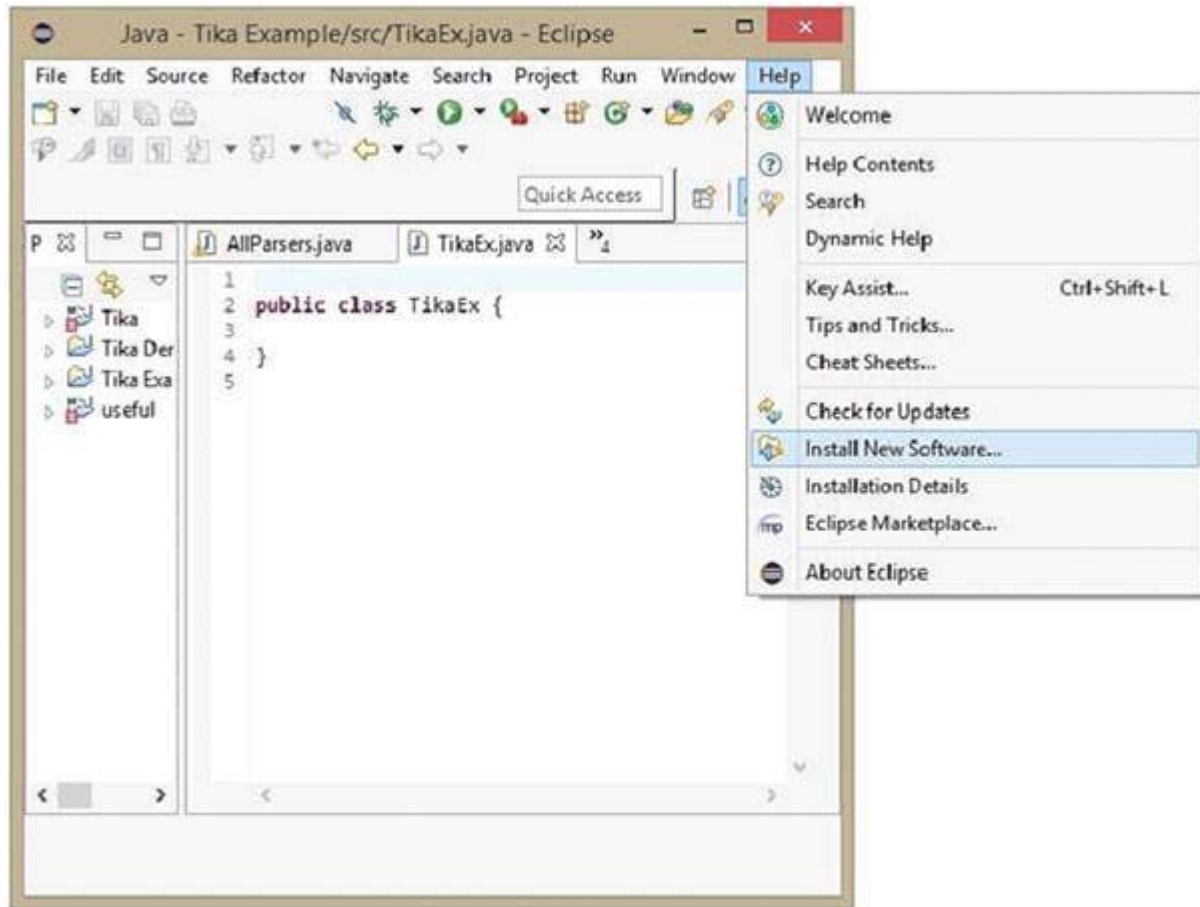
- Open eclipse and create a new project.
- If you do not having Maven in your Eclipse, set it up by following the given steps.
 - Open the [link http://wiki.eclipse.org/M2E_updatesite_and_gittags](http://wiki.eclipse.org/M2E_updatesite_and_gittags). There you will find the m2e plugin releases in a tabular format

release	full version	date	tag	p2 url
1.0	1.0.0 20110607-2117	2011-06-22	releases/1.0/1.0.0 20110607-2117	http://download.eclipse.org/technology/m2e/releases/1.0/1.0.0 20110607-2117
1.0 SR1	1.0.100 20110804-1717	2011-09-23	releases/1.0/1.0.100 20110804-1717	http://download.eclipse.org/technology/m2e/releases/1.0/1.0.100 20110804-1717
1.0 SR2	1.0.200 20111228-1245	2012-02-24	releases/1.0/1.0.200 20111228-1245	http://download.eclipse.org/technology/m2e/releases/1.0/1.0.200 20111228-1245
1.1	1.1.0 20120530-0009	2012-06-27	releases/1.1/1.1.0 20120530-0009	http://download.eclipse.org/technology/m2e/releases/1.1/1.1.0 20120530-0009
1.2	1.2.0 20120903-1050	2012-09-19	releases/1.2/1.2.0 20120903-1050	http://download.eclipse.org/technology/m2e/releases/1.2/1.2.0 20120903-1050
1.3	1.3.0 20130129-0926	2013-02-20	releases/1.3/1.3.0 20130129-0926	http://download.eclipse.org/technology/m2e/releases/1.3/1.3.0 20130129-0926
1.3.1	1.3.1 20130219-1424	2013-02-20	releases/1.3/1.3.1 20130219-1424	http://download.eclipse.org/technology/m2e/releases/1.3/1.3.1 20130219-1424
1.4	1.4.0 20130601-0317	2013-06-26	releases/1.4/1.4.0 20130601-0317	http://download.eclipse.org/technology/m2e/releases/1.4/1.4.0 20130601-0317
1.4.1	1.4.1 20140328-1606	2014-03-31	releases/1.4/1.4.1 20140328-1606	http://download.eclipse.org/technology/m2e/releases/1.4/1.4.1 20140328-1606

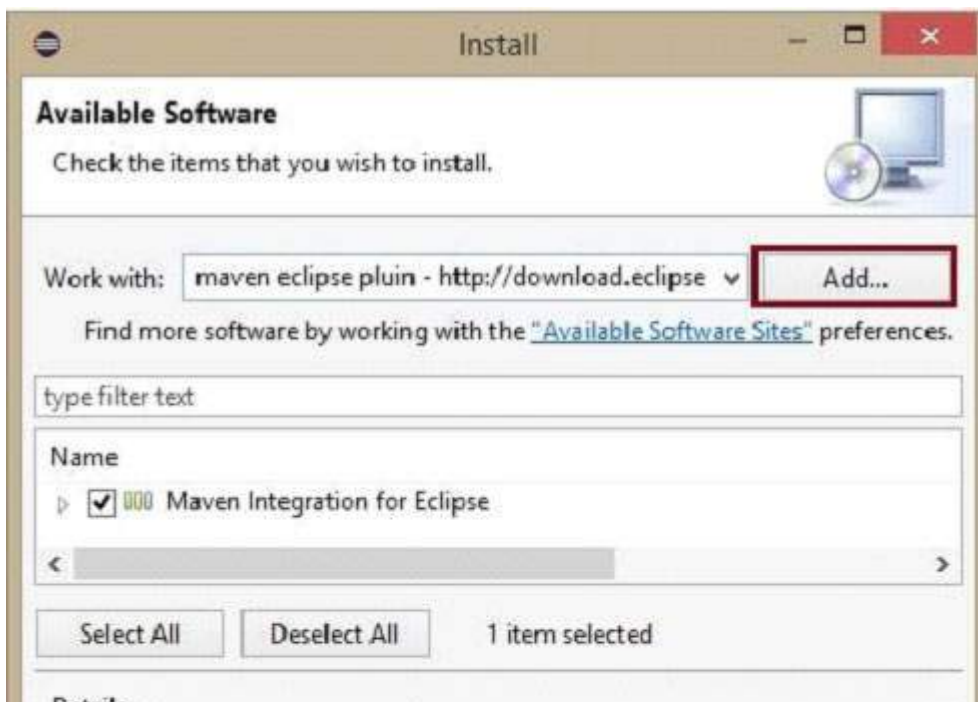
1.5	1.5.0 20140606-0033	2014-06-25 releases/1.5/1.5.0 20140606-0033	http://download.eclipse.org/technology/m2e/releases/1.5/1.5.0 20140606-0033
-----	---------------------	---	---

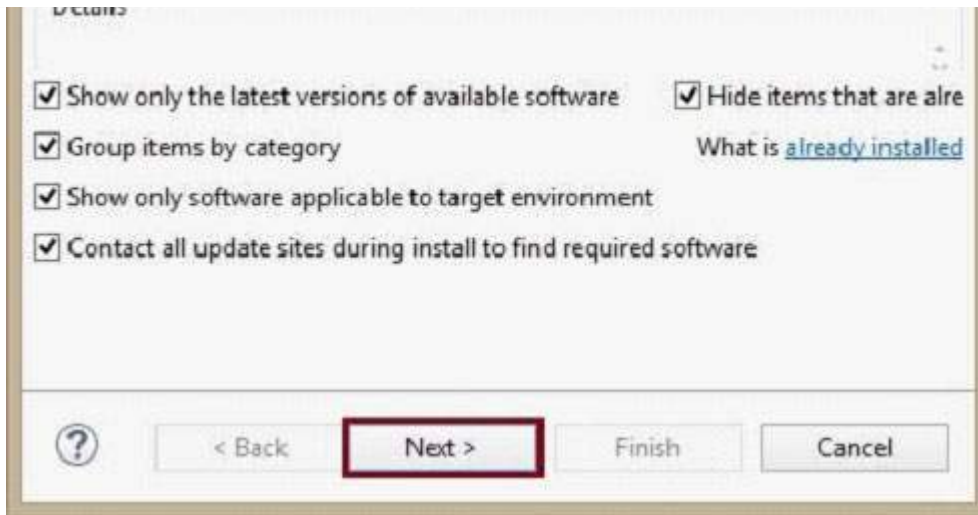
Retrieved from "http://wiki.eclipse.org/index.php?title=M2E_updatesite_and_offings&oldid=307954"
Category:

- Pick the latest version and save the path of the url in p2 url column.
- Now revisit eclipse, in the menu bar, click **Help**, and choose **Install New Software** from the dropdown menu



- Click the **Add** button, type any desired name, as it is optional. Now paste the saved url in the **Location** field.
- A new plugin will be added with the name you have chosen in the previous step, check the checkbox in front of it, and click **Next**.





- Proceed with the installation. Once completed, restart the Eclipse.
- Now right click on the project, and in the **configure** option, select **convert to maven project**.
- A new wizard for creating a new pom appears. Enter the Group Id as org.apache.tika, enter the latest version of Tika, select the **packaging** as jar, and click **Finish**.

The Maven project is successfully installed, and your project is converted into Maven. Now you have to configure the pom.xml file.

Configure the XML File

Get the Tika maven dependency from <http://mvnrepository.com/artifact/org.apache.tika>

Shown below is the complete Maven dependency of Apache Tika.

```
<dependency>
  <groupId>org.apache.Tika</groupId>
  <artifactId>Tika-core</artifactId>
  <version>1.6</version>

  <groupId>org.apache.Tika</groupId>
  <artifactId>Tika-parsers</artifactId>
  <version>1.6</version>

  <groupId>org.apache.Tika</groupId>
  <artifactId>Tika</artifactId>
  <version>1.6</version>

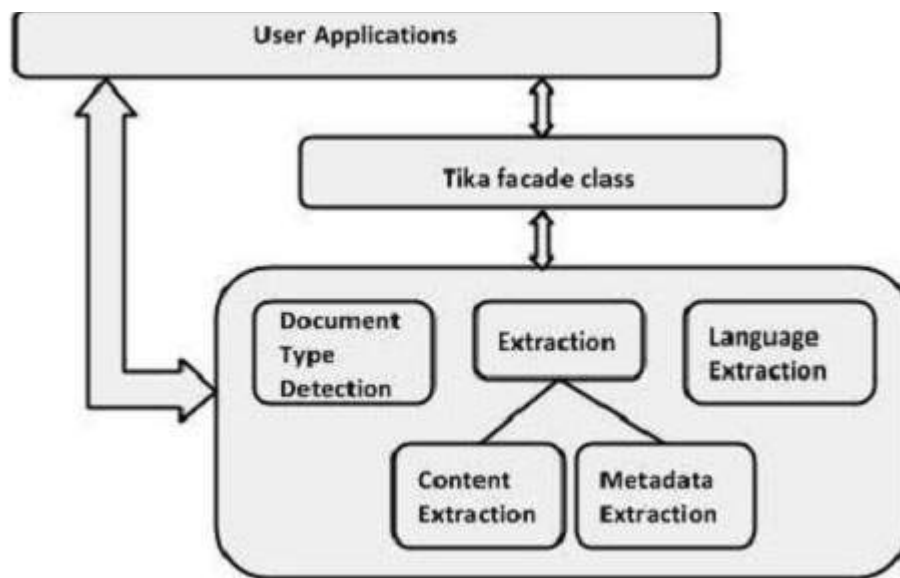
  <groupId>org.apache.Tika</groupId>
  <artifactId>Tika-serialization</artifactId>
  <version>1.6</version>

  <groupId>org.apache.Tika</groupId>
  <artifactId>Tika-app</artifactId>
  <version>1.6</version>

  <groupId>org.apache.Tika</groupId>
  <artifactId>Tika-bundle</artifactId>
  <version>1.6</version>
</dependency>
```

TIKA - REFERENCED API

Users can embed Tika in their applications using the Tika facade class. It has methods to explore all the functionalities of Tika. Since it is a facade class, Tika abstracts the complexity behind its functions. In addition to this, users can also use the various classes of Tika in their applications.



Tika Class *facade*

This is the most prominent class of the Tika library and follows the facade design pattern. Therefore, it abstracts all the internal implementations and provides simple methods to access the Tika functionalities. The following table lists the constructors of this class along with their descriptions.

package : org.apache.tika

class: Tika

S.No.	Constructor and Description
1	<p>Tika</p> <p>Uses default configuration and constructs the Tika class.</p>
2	<p>Tika <i>Detector</i>detector</p> <p>Creates a Tika facade by accepting the detector instance as parameter</p>
3	<p>Tika <i>Detector</i>detector, <i>Parser</i>parser</p> <p>Creates a Tika facade by accepting the detector and parser instances as parameters.</p>
4	<p>Tika <i>Detector</i>detector, <i>Parser</i>parser, <i>Translator</i>translator</p> <p>Creates a Tika facade by accepting the detector, the parser, and the translator instance as parameters.</p>
5	<p>Tika <i>TikaConfig</i>config</p> <p>Creates a Tika facade by accepting the object of the TikaConfig class as parameter.</p>
<h2>Methods and Description</h2>	

The following are the important methods of Tika facade class:

S.No.	Methods and Description
1	<p><u>parseToString Filefile</u></p> <p>This method and all its variants parses the file passed as parameter and returns the extracted text content in the String format. By default, the length of this string parameter is limited.</p>
2	<p>int getMaxStringLength</p> <p>Returns the maximum length of strings returned by the parseToString methods.</p>
3	<p>void setMaxStringLengthintmaxStringLength</p> <p>Sets the maximum length of strings returned by the parseToString methods.</p>
4	<p><u>Reade parse Filefile</u></p> <p>This method and all its variants parses the file passed as parameter and returns the extracted text content in the form of java.io.reader object.</p>
5	<p><u>String detect (InputStream stream, Metadata metadata)</u></p> <p>This method and all its variants accepts an InputStream object and a Metadata object as parameters, detects the type of the given document, and returns the document type name as String object. This method abstracts the detection mechanisms used by Tika.</p>
6	<p><u>String translate (InputStream text, String targetLanguage)</u></p> <p>This method and all its variants accepts the InputStream object and a String representing the language that we want our text to be translated, and translates the given text to the desired language, attempting to auto-detect the source language.</p>

Parser Interface

This is the interface that is implemented by all the parser classes of Tika package.

package : org.apache.tika.parser

Interface : Parser

Methods and Description

The following is the important method of Tika Parser interface:

S.No.	Methods and Description
1	<p>parse InputStreamstream, ContentHandlerhandler, Metadatametadata, ParseContextcontext</p> <p>This method parses the given document into a sequence of XHTML and SAX events.</p>

After parsing, it places the extracted document content in the object of the ContentHandler class and the metadata in the object of the Metadata class.

Metadata Class

This class implements various interfaces such as CreativeCommons, Geographic, HttpHeaders, Message, MSOffice, ClimateForecast, TIFF, TikaMetadataKeys, TikaMimeKeys, Serializable to support various data models. The following tables list the constructors and methods of this class along with their descriptions.

package : org.apache.tika.metadata

class : Metadata

S.No.	Constructor and description
-------	-----------------------------

- | | |
|---|--|
| 1 | Metadata
Constructs a new, empty metadata. |
|---|--|

S.No.	Methods and Description
-------	-------------------------

- | | |
|---|---|
| 1 | add <i>Propertyproperty, Stringvalue</i>
Adds a metadata property/value mapping to a given document. Using this function, we can set the value to a property. |
| 2 | add <i>Stringname, Stringvalue</i>
Adds a metadata property/value mapping to a given document. Using this method, we can set a new name value to the existing metadata of a document. |
| 3 | String get <i>Propertyproperty</i>
Returns the value <i>ifany</i> of the metadata property given. |
| 4 | String get <i>Stringname</i>
Returns the value <i>ifany</i> of the metadata name given. |
| 5 | Date getDate <i>Propertyproperty</i>
Returns the value of Date metadata property. |
| 6 | String[] getValues <i>Propertyproperty</i>
Returns all the values of a metadata property. |
| 7 | String[] getValues <i>Stringname</i> |

Returns all the values of a given metadata name.

8

String[] names

Returns all the names of metadata elements in a metadata object.

9

set Propertyproperty, Datedate

Sets the date value of the given metadata property

10

set Propertyproperty, String[]values

Sets multiple values to a metadata property.

Language Identifier Class

This class identifies the language of the given content. The following tables list the constructors of this class along with their descriptions.

package : org.apache.tika.language

class : Language Identifier

S.No.	Constructor and description
-------	-----------------------------

1

LanguageIdentifier LanguageProfileprofile

Instantiates the language identifier. Here you have to pass a LanguageProfile object as parameter.

2

LanguageIdentifier Stringcontent

This constructor can instantiate a language identifier by passing on a String from text content

S.No.	Methods and Description
-------	-------------------------

1

String getLanguage

Returns the language given to the current LanguageIdentifier object.

TIKA - FILE FORMATS

File Formats Supported by Tika

The following table shows the file formats Tika supports.

File format	Package Library	Class in Tika
-------------	-----------------	---------------

XML	org.apache.tika.parser.xml	XMLParser
HTML	org.apache.tika.parser.html and it uses Tagsoup Library	HtmlParser
MS-Office compound document Ole2 till 2007 ooxml 2007 onwards	org.apache.tika.parser.microsoft org.apache.tika.parser.microsoft.ooxml and it uses Apache Poi library	OfficeParserole2 OOXMLParserooxml
OpenDocument Format openoffice	org.apache.tika.parser.odf	OpenOfficeParser
portable Document Format <i>PDF</i>	org.apache.tika.parser.pdf and this package uses Apache PdfBox library	PDFParser
Electronic Publication Format <i>digitalbooks</i>	org.apache.tika.parser.epub	EpubParser
Rich Text format	org.apache.tika.parser.rtf	RTFParser
Compression and packaging formats	org.apache.tika.parser.pkg and this package uses Common compress library	PackageParser and CompressorParser and its sub-classes
Text format	org.apache.tika.parser.txt	TXTParser
Feed and syndication formats	org.apache.tika.parser.feed	FeedParser
Audio formats	org.apache.tika.parser.audio and org.apache.tika.parser.mp3	AudioParser MidiParser Mp3- for mp3parser
Imageparsers	org.apache.tika.parser.jpeg	JpegParser-for jpeg images
Videoformats	org.apache.tika.parser.mp4 and org.apache.tika.parser.video this parser internally uses Simple Algorithm to parse flash video formats	Mp4parser FlvParser
java class files and jar files	org.apache.tika.parser.asm	ClassParser CompressorParser
Mobxformat <i>emailmessages</i>	org.apache.tika.parser.mbox	MobXParser
Cad formats	org.apache.tika.parser.dwg	DWGParser
FontFormats	org.apache.tika.parser.font	TrueTypeParser
executable programs and libraries	org.apache.tika.parser.executable	ExecutableParser
TIKA - DOCUMENT TYPE DETECTION		

MIME Standards

Multipurpose Internet Mail Extensions *MIME* standards are the best available standards for identifying document types. The knowledge of these standards helps the browser during internal interactions.

Whenever the browser encounters a media file, it chooses a compatible software available with it to display its contents. In case it does not have any suitable application to run a particular media file, it recommends the user to get the suitable plugin software for it.

Type Detection in Tika

Tika supports all the Internet media document types provided in MIME. Whenever a file is passed through Tika, it detects the file and its document type. To detect media types, Tika internally uses the following mechanisms.

File extensions

Checking the file extensions is the simplest and most-widely used method to detect the format of a file. Many applications and operating systems provide support for these extensions. Shown below are the extension of a few known file types.

File name	Extention
image	.jpg
audio	.mp3
java archive file	.jar
java class file	.class

Content-type hints

Whenever you retrieve a file from a database or attach it to another document, you may lose the file's name or extension. In such cases, the metadata supplied with the file is used to detect the file extension.

Magic bytes

Observing the raw bytes of a file, you can find some unique character patterns for each file. Some files have special byte prefixes called **magic bytes** that are specially made and included in a file for the purpose of identifying the file type.

For example, you can find CA FE BA BE *hexadecimalformat* in a java file and %PDF *ASCIIformat* in a pdf file. Tika uses this information to identify the media type of a file.

Character encodings

Files with plain text are encoded using different types of character encoding. The main challenge here is to identify the type of character encoding used in the files. Tika follows character encoding techniques like **Bom markers** and **Byte Frequencies** to identify the encoding system used by the plain text content.

Xml root characters

To detect XML documents, Tika parses the xml documents and extracts the information such as root elements, namespaces, and referenced schemas from where the true media type of the files can be found.

Type detection using Facade class

The **detect** method of facade class is used to detect the document type. This method accepts a file as input. Shown below is an example program for document type detection with Tika facade class.

```
import java.io.File;
```

```

import java.util.Scanner;

import org.apache.tika.Tika;

Public class TikaTypedetection {

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

        Scanner scanner = new Scanner(System.in);
        String filepath = scanner.nextLine();
        System.out.println(filepath);

        //creating file object
        File file = new File(filepath);

        //using tika facade class
        Tika tika = new Tika();

        //detecting the file type using detect method
        String filetype = tika.detect(file);
        System.out.println(filetype);
    }
}

```

Save the above code as TypeDetection.java and run it from the command prompt using the following commands:

```

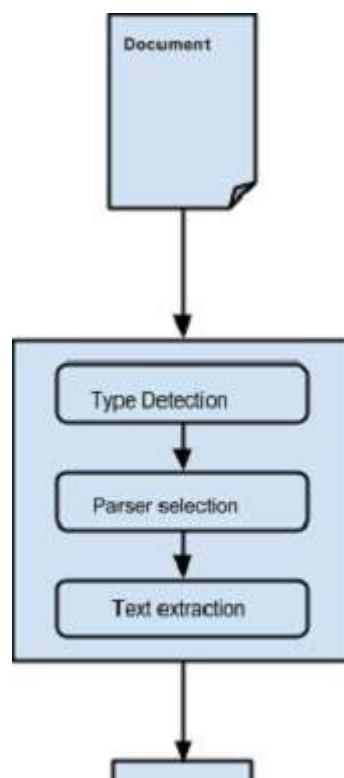
javac TypeDetection.java
java TypeDetection
enter path of your file
c:\tika example\ring.mp3
audio/mpeg

```

TIKA - CONTENT EXTRACTION

Tika uses various parser libraries to extract content from given parsers. It chooses the right parser for extracting the given document type.

For parsing documents, the parseToString method of Tika facade class is generally used. Shown below are the steps involved in the parsing process and these are abstracted by the Tika ParseToString method.





Abstracting the parsing process:

- Initially when we pass a document to Tika, it uses a suitable type detection mechanism available with it and detects the document type.
- Once the document type is known, it chooses a suitable parser from its parser repository. The parser repository contains classes that make use of external libraries.
- Then the document is passed to choose the parser which will parse the content, extract the text, and also throw exceptions for unreadable formats.

Content Extraction using Tika

Given below is the program for extracting text from a file using Tika facade class:

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

import org.apache.tika.Tika;
import org.apache.tika.exception.TikaException;
import org.xml.sax.SAXException;

public class TikaExtraction {
    public static void main(final String[] args) throws IOException,
        SAXException, TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("enter path of your file");
        String filepath = scanner.nextLine();

        //creating the file object
        File file = new File(filepath);

        //using tika facade
        Tika tika = new Tika();
        String filecontent = tika.parseToString(file);
        System.out.println("Extracted Content: " + filecontent);
    }
}
```

Save the above code as TikaExtraction.java and run it from the command prompt:

```
javac TikaExtraction.java
java TikaExtraction
```

Note : Here you have to pass the full path of your file. For example, we are passing the location of a sample text document with the following content.

```
Hi students welcome to tutorialspoint
```

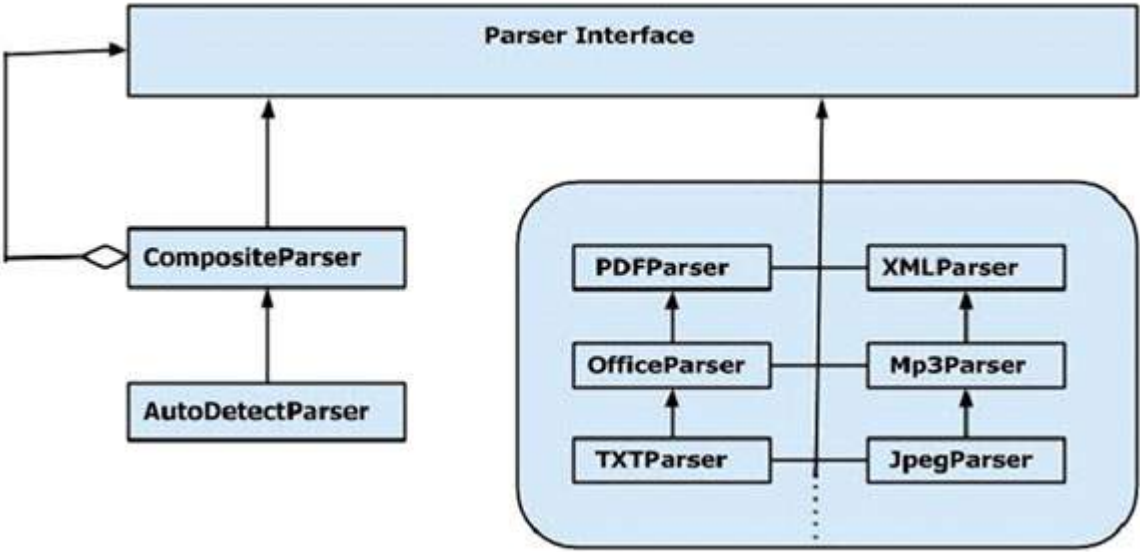
It gives you the following output:

```
enter path of your file
c:\Tika example\sample.txt
```

```
Extracted Content: Hi students welcome to tutorialspoint
```

Content Extraction using Parser Interface

The parser package of Tika provides several interfaces and classes using which we can parse a text document. Given below is the block diagram of the org.apache.tika.parser package.



There are several parser classes available, e.g., pdf parser, Mp3Passer, OfficeParser, etc., to parse respective documents individually. All these classes implement the parser interface.

CompositeParser

The given diagram shows Tika’s general-purpose parser classes **CompositeParser** and **AutoDetectParser**. Since the CompositeParser class follows composite design pattern, you can use a group of parser instances as a single parser. The CompositeParser class also allows access to all the classes that implement the parser interface.

AutoDetectParser

This is a subclass of CompositeParser and it provides automatic type detection. Using this functionality, the AutoDetectParser automatically sends the incoming documents to the appropriate parser classes using the composite methodology.

parse method

Along with parseToString, you can also use the parse method of the parser Interface. The prototype of this method is shown below.

```
parse(InputStream stream, ContentHandler handler, Metadata metadata,
ParseContext context)
```

The following table lists the four objects it accepts as parameters.

S.No.	Object and Description
1	InputStream stream Any Inputstream object that contains the content of the file
2	ContentHandler handler Tika passes the document as XHTML content to this handler, thereafter the document is processed using SAX API. It provides efficient post-processing of the contents in a document.

3

Metadata metadata

The metadata object is used both as a source and a target of document metadata.

4

ParseContext context

This object is used in cases where the client application wants to customize the parsing process.

Example:

Given below is an example that shows how the parse method is used.

Step 1:

To use the parse method of the parser interface, instantiate any of the classes providing the implementation for this interface.

There are individual parser classes such as PDFParser, OfficeParser, XMLParser, etc. You can use any of these individual document parsers. Alternatively, you can use either CompositeParser or AutoDetectParser that uses all the parser classes internally and extracts the contents of a document using a suitable parser.

```
Parser parser = new AutoDetectParser();  
(or)  
Parser parser = new CompositeParser();  
(or)  
object of any individual parsers given in Tika Library
```

Step 2:

Create a handler class object. Given below are the three content handlers:

S.No.	Class and Description
-------	-----------------------

1

BodyContentHandler

This class picks the body part of the XHTML output and writes that content to the output writer or output stream. Then it redirects the XHTML content to another content handler instance.

2

LinkContentHandler

This class detects and picks all the H-Ref tags of the XHTML document and forwards those for the use of tools like web crawlers.

3

TeeContentHandler

This class helps in using multiple tools simultaneously.

Since our target is to extract the text contents from a document, instantiate BodyContentHandler as shown below:


```
BodyContentHandler handler = new BodyContentHandler( );
```

Step 3:

Create the Metadata object as shown below:

```
Metadata metadata = new Metadata();
```

Step 4:

Create any of the input stream objects, and pass your file that should be extracted to it.

FileInputStream

Instantiate a file object by passing the file path as parameter and pass this object to the FileInputStream class constructor.

Note : The path passed to the file object should not contain spaces.

The problem with these input stream classes is that they don't support random access reads, which is required to process some file formats efficiently. To resolve this problem, Tika provides TikaInputStream.

```
File file=new File(filepath)
FileInputStream inputstream=new FileInputStream(file);
(or)
InputStream stream = TikaInputStream.get(new File(filename));
```

Step 5:

Create a parse context object as shown below:

```
ParseContext context =new ParseContext();
```

Step 6:

Instantiate the parser object, invoke the parse method, and pass all the objects required, as shown in the prototype below:

```
parser.parse(inputstream, handler, metadata, context);
```

Given below is the program for content extraction using the parser interface:

```
package ContentExtraction;

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

import java.util.Scanner;

import org.apache.Tika.exception.TikaException;
import org.apache.Tika.metadata.Metadata;
import org.apache.Tika.parser.AutoDetectParser;
import org.apache.Tika.parser.ParseContext;
import org.apache.Tika.parser.Parser;
import org.apache.Tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class ParserExtraction {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {
```

```

Scanner scanner = new Scanner(System.in);
System.out.println("enter path of your file");
String filepath = scanner.nextLine();

//creating file object
File file = new File(filepath);

//parse method parameters
Parser parser = new AutoDetectParser();
BodyContentHandler handler = new BodyContentHandler();
Metadata metadata = new Metadata();
FileInputStream inputStream=new FileInputStream(file);
ParseContext context =new ParseContext();

parser.parse(inputStream, handler, metadata, context);
System.out.println(handler.toString());
}
}

```

Save the above code as ParserExtraction.java and run it from the command prompt:

```

javac ParserExtraction.java
java ParserExtraction

```

Note : Here you have to pass your file path. For example, we are passing the location of a sample text document that contains the following content.

```
Hi students welcome to tutorialspoint
```

It gives the following output:

```

Enter path of your file
c:\Tika example\sample.txt
Extracted Content: Hi students welcome to tutorialspoint

```

TIKA - METADATA EXTRACTION

Besides content, Tika also extracts the metadata from a file. Metadata is nothing but the additional information supplied with a file. If we consider an audio file, the artist name, album name, title comes under metadata.

XMP Standards

The Extensible Metadata Platform *XMP* is a standard for processing and storing information related to the content of a file. It was created by [Adobe Systems Inc.](#) XMP provides standards for defining, creating, and processing of [metadata](#). You can embed this standard into several file formats such as [PDF](#), [JPEG](#), [GIF](#), [jpg](#), [HTML](#), etc.

Property Class

Tika uses the Property class to follow XMP property definition. It provides the [PropertyType](#) and [ValueType](#) enums to capture the name and value of a metadata.

Metadata Class

This class implements various interfaces such as [ClimateForecast](#), [CativeCommons](#), [Geographic](#), [TIFF](#) etc. to provide support for various metadata models. In addition, this class provides various methods to extract the content from a file.

Metadata Names

We can extract the list of all metadata names of a file from its metadata object using the method [names](#). It returns all the names as a string array. Using the name of the metadata, we can get the value using the **get** method. It takes a metadata name and returns a value associated with it.

```
String[] metadaNames = metadata.names();

String value = metadata.get(name);
```

Extracting Metadata using Parse Method

Whenever we parse a file using parse, we pass an empty metadata object as one of the parameters. This method extracts the metadata of the given file *if that file contains any*, and places them in the metadata object. Therefore, after parsing the file using parse, we can extract the metadata from that object.

```
Parser parser = new AutoDetectParser();
BodyContentHandler handler = new BodyContentHandler();
Metadata metadata = new Metadata(); //empty metadata object
FileInputStream inputstream = new FileInputStream(file);
ParseContext context = new ParseContext();
parser.parse(inputstream, handler, metadata, context);
// now this metadata object contains the extracted metadata of the given file.
metadata.names();
```

Given below is the complete program to extract metadata from a text file.

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

import java.util.Scanner;
import java.util.StringTokenizer;

import org.apache.Tika.exception.TikaException;
import org.apache.Tika.metadata.Metadata;
import org.apache.Tika.parser.AutoDetectParser;
import org.apache.Tika.parser.ParseContext;
import org.apache.Tika.parser.Parser;
import org.apache.Tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class GetMetadata {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("enter path of your file");
        String filepath = scanner.nextLine();

        //creating file object
        File file = new File(filepath);
        Parser parser = new AutoDetectParser();
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputstream = new FileInputStream(file);
        ParseContext context = new ParseContext();
        parser.parse(inputstream, handler, metadata, context);
        System.out.println(handler.toString());

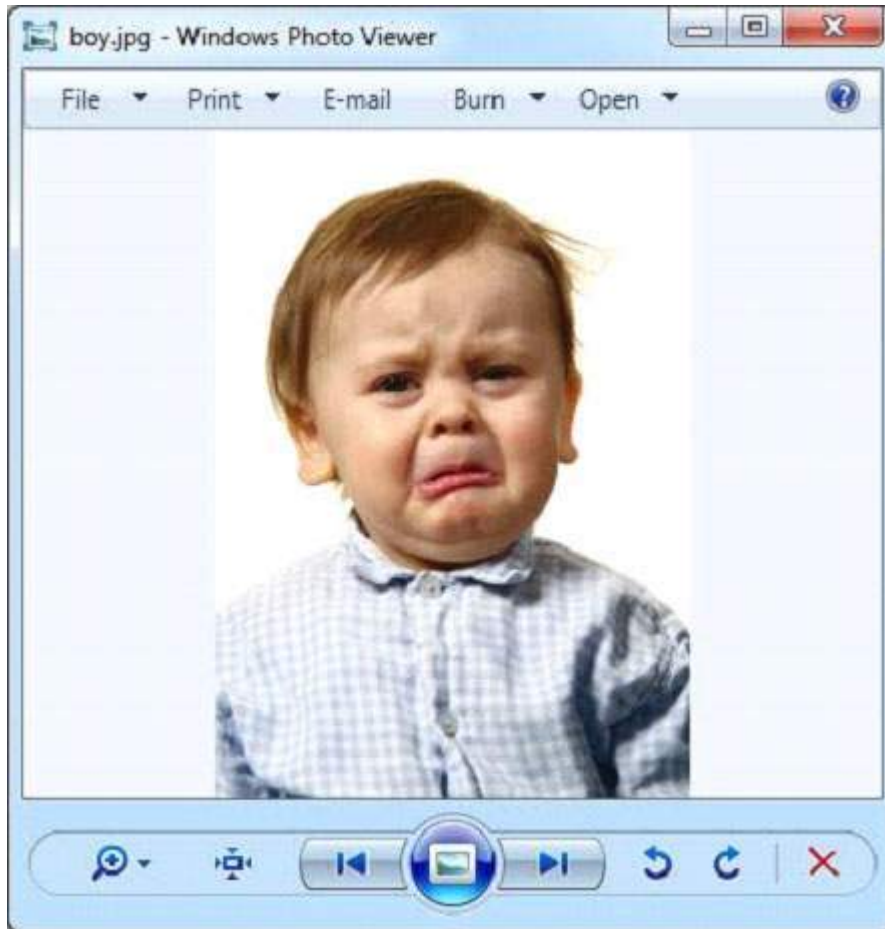
        //getting the list of all meta data elements
        String[] metadataNames = metadata.names();

        for(String name : metadataNames) {
            System.out.println(name + ": " + metadata.get(name));
        }
    }
}
```

Save the above code as GetMetadata.java and run it from the command prompt using the following commands:

```
javac GetMetadata.java
java GetMetadata
```

Note : Here you have to pass your file path. For example, we are passing the location of the following image.



It gives the following output:

```
enter path of your file
E:\office\Tika\stand alone\parsers\boy.jpg

Resolution Units: inch
Compression Type: Baseline
Data Precision: 8 bits
Number of Components: 3
tiff:ImageLength: 1000
Component 2: Cb component: Quantization table 1, Sampling factors 1 horiz/1 vert
Component 1: Y component: Quantization table 0, Sampling factors 1 horiz/1 vert
Image Height: 1000 pixels
X Resolution: 300 dots
Original Transmission Reference:
53616c7465645f5fd22a84941585d89cc735d889c9d5ac58a01faf2c92ee3c6f9bcb38359bbe1eef
Image Width: 714 pixels
IPTC-NAA record: 92 bytes binary data
Component 3: Cr component: Quantization table 1, Sampling factors 1 horiz/1 vert
tiff:BitsPerSample: 8
Application Record Version: 4
tiff:ImageWidth: 714
Content-Type: image/jpeg
Y Resolution: 300 dots
```

We can also get our desired metadata values.

Adding New Metadata Values

We can add new metadata values using the add method of the metadata class. Given below is the syntax of this method. Here we are adding the author name.

```
metadata.add("author", "Tutorials point");
```

The Metadata class has predefined properties including the properties inherited from classes like [ClimateForecast](#), [CativeCommons](#), [Geographic](#), etc., to support various data models. Shown below is the usage of the SOFTWARE data type inherited from the TIFF interface implemented by Tika to follow XMP metadata standards for TIFF image formats.

```
metadata.add(Metadata.SOFTWARE, "ms paint");
```

Given below is the complete program that demonstrates how to add metadata values to a given file. Here the list of the metadata elements is displayed in the output so that you can observe the change in the list after adding new values.

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

import java.util.Arrays;
import java.util.Scanner;

import org.apache.Tika.exception.TikaException;
import org.apache.Tika.metadata.Metadata;
import org.apache.Tika.parser.AutoDetectParser;
import org.apache.Tika.parser.ParseContext;
import org.apache.Tika.parser.Parser;
import org.apache.Tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class AddMetadata {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("enter path of your file");

        String filepath = scanner.nextLine();
        File file = new File(filepath);
        Parser parser = new AutoDetectParser();
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputstream = new FileInputStream(file);
        ParseContext context = new ParseContext();

        parser.parse(inputstream, handler, metadata, context);

        //list of meta data elements before adding new elements
        System.out.println("metadata elements of the given file " +
Arrays.toString(metadata.names()));
        System.out.println("enter the number of metadata name value pairs to be added ");
        int number = scanner.nextInt();

        for(int i=0; i<number i++) {
            System.out.println("enter metadata" + (i+1) + "name:");
            String name = scanner.next();
            System.out.println("enter metadata" + (i+1) + "value:");
            String value = scanner.next();

            metadata.add(name, value);
            metadata.add(name, value);
        }
    }
}
```

```

        System.out.println(" metadata name value pair is successfully added");

        //printing all the meta data elements after adding new elements
        System.out.println("Here is the list of all the metadata elements  after adding new
elements ");
        System.out.println( Arrays.toString(metadata.names()));
    }
}

```

Save the above code as AddMetadata.java class and run it from the command prompt:

```

javac AddMetadata .java
java AddMetadata

```

Note : Here you have to pass your file path. For example, we are passing the location of a sample text document with the following content:

```

Hi students welcome to tutorialspoint

```

It gives the following output:

```

enter path of your file
E:\office\Tika\stand alone\parsers\one.txt
metadata elements of the given file :[Content-Encoding, Content-Type]
enter the number of metadata name value pairs to be added
1
enter metadata1name:
Author
enter metadata1value:
Tutorials point
metadata name value pair is successfully added
Here is the list of all the metadata elements  after adding new elements
[Content-Encoding, Author, Content-Type]

```

Setting Values to Existing Metadata Elements

You can set values to the existing metadata elements using the set method. The syntax of setting the date property using the set method is as follows:

```

metadata.set(Metadata.DATE, new Date());

```

You can also set multiple values to the properties using the set method. The syntax of setting multiple values to the Author property using the set method is as follows:

```

metadata.set(Metadata.AUTHOR, "ram ,raheem ,robin ");

```

Given below is the complete program demonstrating the set method.

```

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

import java.util.Arrays;
import java.util.Date;
import java.util.Scanner;

import org.apache.Tika.exception.TikaException;
import org.apache.Tika.metadata.Metadata;
import org.apache.Tika.parser.AutoDetectParser;
import org.apache.Tika.parser.ParseContext;
import org.apache.Tika.parser.Parser;
import org.apache.Tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

```



```

public class SetMetadata {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        //accepting file path from user
        Scanner scanner = new Scanner(System.in);
        System.out.println("enter path of your file");
        String filepath = scanner.nextLine();

        //parameters of parse() method
        File file=new File(filepath);
        Parser parser = new AutoDetectParser();
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputStream=new FileInputStream(file);
        ParseContext context =new ParseContext();
        parser.parse(inputStream, handler, metadata, context);

        //list of meta data elements elements
        System.out.println( " metadata elements and values of the given file :");
        String[] metadataNamesb4 = metadata.names();

        for(String name : metadataNamesb4) {
            System.out.println(name + ": " + metadata.get(name));
        }

        //setting date meta data
        metadata.set(Metadata.DATE, new Date());

        //setting multiple values to author property
        metadata.set(Metadata.AUTHOR, "ram, raheem, robin ");

        //printing all the meta data elements with new elements
        System.out.println("List of all the metadata elements with new elements ");
        String[] metadataNamesafter = metadata.names();

        for(String name : metadataNamesafter) {
            System.out.println(name + ": " + metadata.get(name));
        }
    }
}

```

Save the above code as SetMetadata.java and run it from the command prompt:

```

javac SetMetadata.java
java SetMetadata

```

Note : Here you have to pass your file path. For example, we are passing the location of a sample text document with the following content:

```
Hi students welcome to tutorialspoint
```

In the output, you can observe the newly added metadata elements.

```

enter path of your file
E:\office\Tika\stand alone\parsers\one.txt
metadata elements and values of the given file :
Content-Encoding: ISO-8859-1
Content-Type: text/plain; charset=ISO-8859-1
Here is the list of all the metadata elements  after adding new elements
date: 2014-09-24T07:01:32Z
Content-Encoding: ISO-8859-1
Author: ram, raheem, robin
Content-Type: text/plain; charset=ISO-8859-1

```

Need for Language Detection

For classification of documents based on the language they are written in a multilingual website, a language detection tool is needed. This tool should accept documents without language annotation *metadata* and add that information in the metadata of the document by detecting the language.

Algorithms for Profiling Corpus

What is Corpus?

To detect the language of a document, a language profile is constructed and compared with the profile of the known languages. The text set of these known languages is known as a **corpus**.

A corpus is a collection of texts of a written language that explains how the language is used in real situations.

The corpus is developed from books, transcripts, and other data resources like the Internet. The accuracy of the corpus depends upon the profiling algorithm we use to frame the corpus.

What are Profiling Algorithms?

The common way of detecting languages is by using dictionaries. The words used in a given piece of text will be matched with those that are in the dictionaries.

A list of common words used in a language will be the most simple and effective corpus for detecting a particular language, for example, articles **a, an, the** in English.

Using Word Sets as Corpus

Using word sets, a simple algorithm is framed to find the distance between two corpora, which will be equal to the sum of differences between the frequencies of matching words.

Such algorithms suffer from the following problems:

- Since the frequency of matching words is very less, the algorithm cannot efficiently work with small texts having few sentences. It needs a lot of text for accurate match.
- It cannot detect word boundaries for languages having compound sentences, and those having no word dividers like spaces or punctuation marks.

Due to these difficulties in using word sets as corpus, individual characters or character groups are considered.

Using Character Sets as Corpus

Since the characters that are commonly used in a language are finite in number, it is easy to apply an algorithm based on word frequencies rather than characters. This algorithm works even better in case of certain character sets used in one or very few languages.

This algorithm suffers from the following drawbacks:

- It is difficult to differentiate two languages having similar character frequencies.
- There is no specific tool or algorithm to specifically identify a language with the help of *ascorpus* the character set used by multiple languages.

N-gram Algorithm

The drawbacks stated above gave rise to a new approach of using character sequences of a given length for profiling corpus. Such sequence of characters are called as N-grams in general, where N represents the length of the character sequence.

- N-gram algorithm is an effective approach for language detection, especially in case of European languages like English.
- This algorithm works fine with short texts.
- Though there are advanced language profiling algorithms to detect multiple languages in a multilingual document having more attractive features, Tika uses the 3-grams algorithm, as it is suitable in most practical situations.

Language Detection in Tika

Among all the 184 standard languages standardized by ISO 639-1, Tika can detect 18 languages. Language detection in Tika is done using the **getLanguage** method of the **LanguageIdentifier** class. This method returns the code name of the language in String format. Given below is the list of the 18 language-code pairs detected by Tika:

da—Danish	de—German	et—Estonian	el—Greek
en—English	es—Spanish	fi—Finnish	fr—French
hu—Hungarian	is—Icelandic	it—Italian	nl—Dutch
no—Norwegian	pl—Polish	pt—Portuguese	ru—Russian
sv—Swedish	th—Thai		

While instantiating the **LanguageIdentifier** class, you should pass the String format of the content to be extracted, or a **LanguageProfile** class object.

```
LanguageIdentifier object=new LanguageIdentifier("this is english");
```

Given below is the example program for Language detection in Tika.

```
import java.io.IOException;
import java.util.Scanner;

import org.apache.Tika.exception.TikaException;
import org.apache.Tika.language.LanguageIdentifier;
import org.xml.sax.SAXException;

public class LanguageDetection {

    public static void main(String args[])throws IOException, SAXException, TikaException
    {

        Scanner scanner = new Scanner(System.in);
        System.out.println("enter the text content of any standard language");
        String content = scanner.nextLine();
        LanguageIdentifier identifier = new LanguageIdentifier(content);
        String language = identifier.getLanguage();
        System.out.println("Language of the given content is:" + language);
    }
}
```

Save the above code as **LanguageDetection.java** and run it from the command prompt using the following commands:

```
javac LanguageDetection.java
java LanguageDetection
```

It gives the following output:

```
enter the text content of any standard language
```

```
this is english
Language of the given content is:en
```

Language Detection of a Document

To detect the language of a given document, you have to parse it using the parse method. The parse method parses the content and stores it in the handler object, which was passed to it as one of the arguments. Pass the String format of the handler object to the constructor of the **LanguageIdentifier** class as shown below:

```
parser.parse(inputStream, handler, metadata, context);
LanguageIdentifier object = new LanguageIdentifier(handler.toString());
```

Given below is the complete program that demonstrates how to detect the language of a given document:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Scanner;

import org.apache.Tika.Tika;
import org.apache.Tika.exception.TikaException;
import org.apache.Tika.metadata.Metadata;
import org.apache.Tika.parser.AutoDetectParser;
import org.apache.Tika.parser.ParseContext;
import org.apache.Tika.parser.Parser;
import org.apache.Tika.sax.BodyContentHandler;
import org.apache.Tika.language.*;

import org.xml.sax.SAXException;

public class DocumentLanguageDetection {
    public static void main(final String[] args) throws IOException, SAXException,
    TikaException {

        Scanner scanner=new Scanner(System.in);
        System.out.println("enter the path of the file ");
        String filepath=scanner.nextLine();
        File file=new File(filepath);
        Tika Tika = new Tika();

        //detecting the file type
        Parser parser = new AutoDetectParser();
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream content=new FileInputStream(file);
        parser.parse(content, handler, metadata, new ParseContext());
        LanguageIdentifier object=new LanguageIdentifier(handler.toString());
        System.out.println("name :"+ object.getLanguage());
    }
}
```

Save the above code as SetMetadata.java and run it from the command prompt:

```
javac SetMetadata.java
java SetMetadata
```

Note : Here you have to pass your file path. For example, we are passing the location of a sample text document with the following content:

```
Hi students welcome to tutorialspoint
```

It gives the following output:

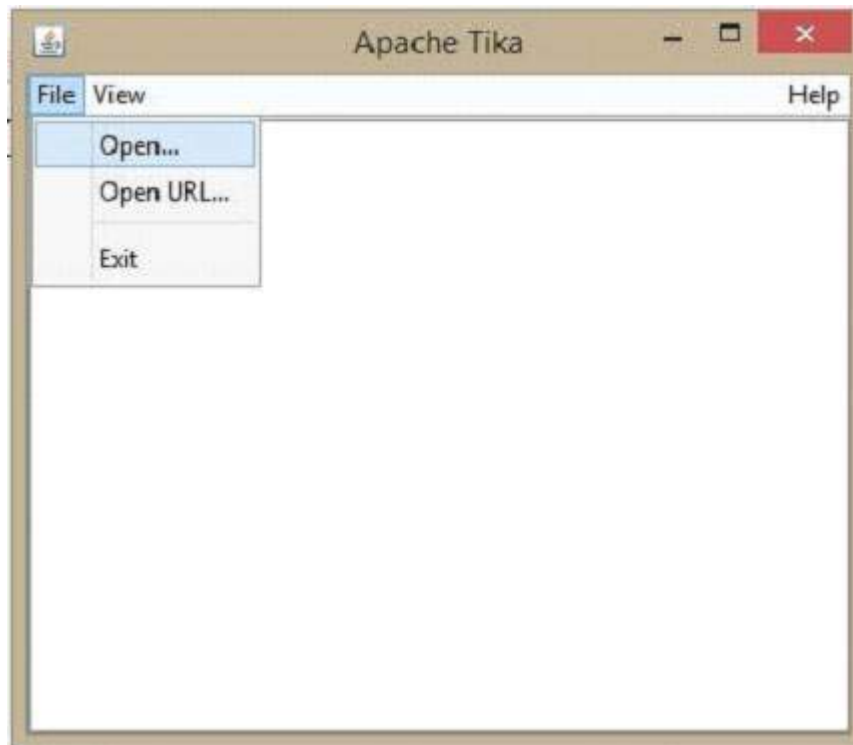
```
enter the path of the file
E:\office\Tika\stand alone\parsers\one.txt
name :en
```

Along with the Tika jar, Tika provides a Graphical User Interface application *GUI* and a Command Line Interface *CLI* application. You can execute a Tika application from the command prompt too like other java applications.

TIKA - GUI

Graphical User Interface *GUI*

- Tika provides a jar file along with its source code in the following link <http://tika.apache.org/download.html>.
- Download both the files, set the classpath for the jar file.
- Extract the source code zip folder, open the tika-app folder.
- In the extracted folder at “tika-1.6\tika-app\src\main\java\org\apache\Tika\gui” you will see two class files: ParsingTransferHandler.java and TikaGUI.java.
- Compile both the class files and execute the TikaGUI.java class file, it opens the following window.



Let us now see how to make use of the Tika GUI.

On the GUI, click open, browse and select a file that is to be extracted, or drag it onto the whitespace of the window.

Tika extracts the content of the files and displays it in five different formats, viz. metadata, formatted text, plain text, main content, and structured text. You can choose any of the format you want.

In the same way, you will also find the CLI class in the “tika-1.6\tika-app\src\main\java\org\apache\tika\cli” folder.

The following illustration shows what Tika can do. When we drop the image on the GUI, Tika extracts and displays its metadata.





TIKA - EXTRACTING PDF

Given below is the program to extract content and metadata from a PDF.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Scanner;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.pdf.PDFParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class PdfParse {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        Scanner scanner =new Scanner(System.in);
        System.out.println("enter path of your pdf file");
        String fileName=scanner.nextLine();
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputStream = new FileInputStream(new File(fileName));
        ParseContext pcontext = new ParseContext();

        //parsing the document using PDF parser
        PDFParser pdfparser = new PDFParser();
        pdfparser.parse(inputStream, handler, metadata,pcontext);

        //getting the content of the document
        System.out.println("Contents of the PDF :" + handler.toString());

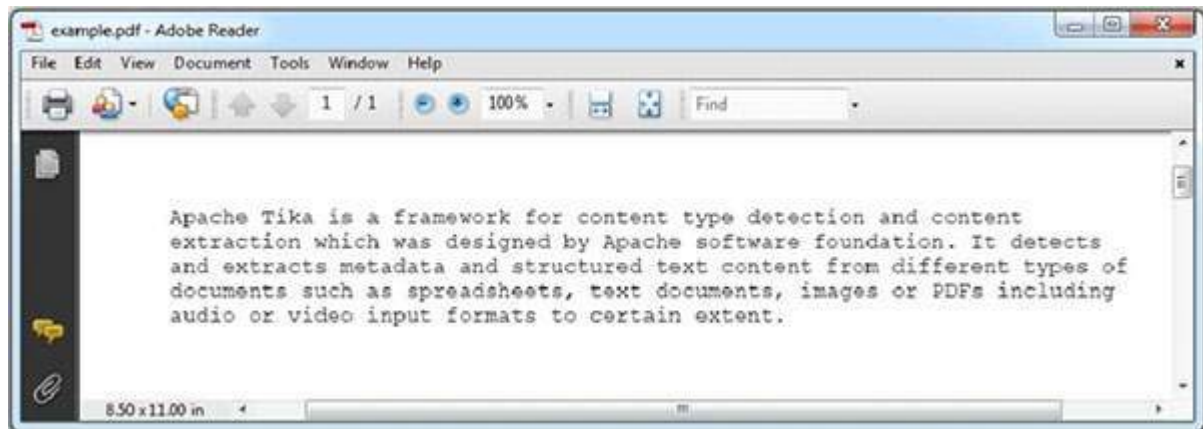
        //getting metadata of the document
        System.out.println("Metadata of the PDF:");
        String[] metadataNames = metadata.names();

        for(String name : metadataNames) {
            System.out.println(name+ " : " + metadata.get(name));
        }
    }
}
```

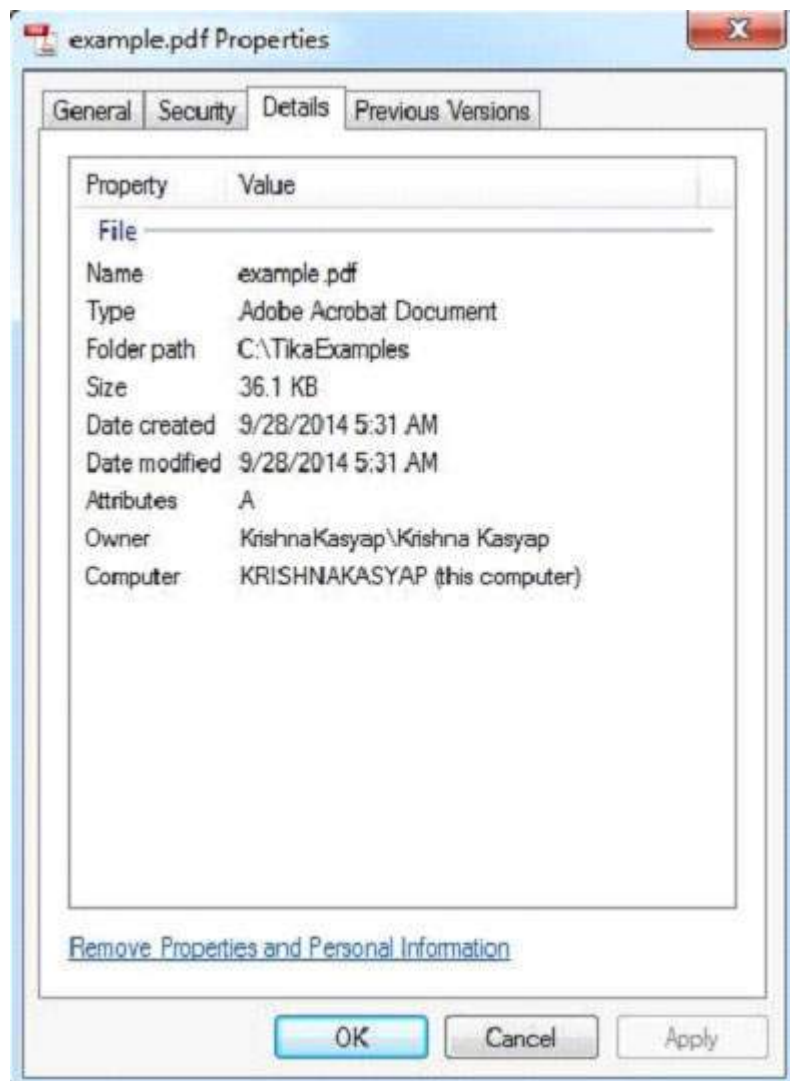
Save the above code as **PdfParse.java**, and compile it from the command prompt by using the following commands:


```
javac PdfParse.java
java PdfParse
```

The application will ask you to enter the path of the pdf document that is to be parsed. Here we are passing the following PDF Document:



The PDF we are passing has the following properties:



After compiling the program, enter the full path of the PDF that is to be extracted.

```
enter path of your pdf file
C:\TikaExamples\example.pdf
```

Output:

```
Contents of the PDF:
```

Apache Tika is a framework for content type detection and content extraction which was designed by Apache software foundation. It detects and extracts metadata and structured text content from different types of documents such as spreadsheets, text documents, images or PDFs including audio or video input formats to certain extent.

Metadata of the PDF:

```
dcterms:modified :      2014-09-28T12:31:16Z
meta:creation-date :    2014-09-28T12:31:16Z
meta:save-date :       2014-09-28T12:31:16Z
dc:creator :           Krishna Kasyap
pdf:PDFVersion :       1.5
Last-Modified :        2014-09-28T12:31:16Z
Author :               Krishna Kasyap
dcterms:created :      2014-09-28T12:31:16Z
date :                 2014-09-28T12:31:16Z
modified :             2014-09-28T12:31:16Z
creator :              Krishna Kasyap
xmpTPg:NPages :        1
Creation-Date :        2014-09-28T12:31:16Z
pdf:encrypted :        false
meta:author :          Krishna Kasyap
created :              Sun Sep 28 05:31:16 PDT 2014
dc:format :            application/pdf; version=1.5
producer :             Microsoft® Word 2013
Content-Type :         application/pdf
xmp:CreatorTool :      Microsoft® Word 2013
Last-Save-Date :      2014-09-28T12:31:16Z
```

TIKA - EXTRACTING ODF

Given below is the program to extract content and metadata from Open Office Document Format ODF.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Scanner;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.odf.OpenDocumentParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class OpenDocumentParse {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("enter path of your odf file");
        String fileName=scanner.nextLine();

        //detecting the file type
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputstream = new FileInputStream(new File(fileName));
        ParseContext pcontext = new ParseContext();

        //Open Document Parser
        OpenDocumentParser openofficeparser = new OpenDocumentParser();
        openofficeparser.parse(inputstream, handler, metadata, pcontext);
        System.out.println("Contents of the document:" +handler.toString());
        System.out.println("Metadata of the document:");
```

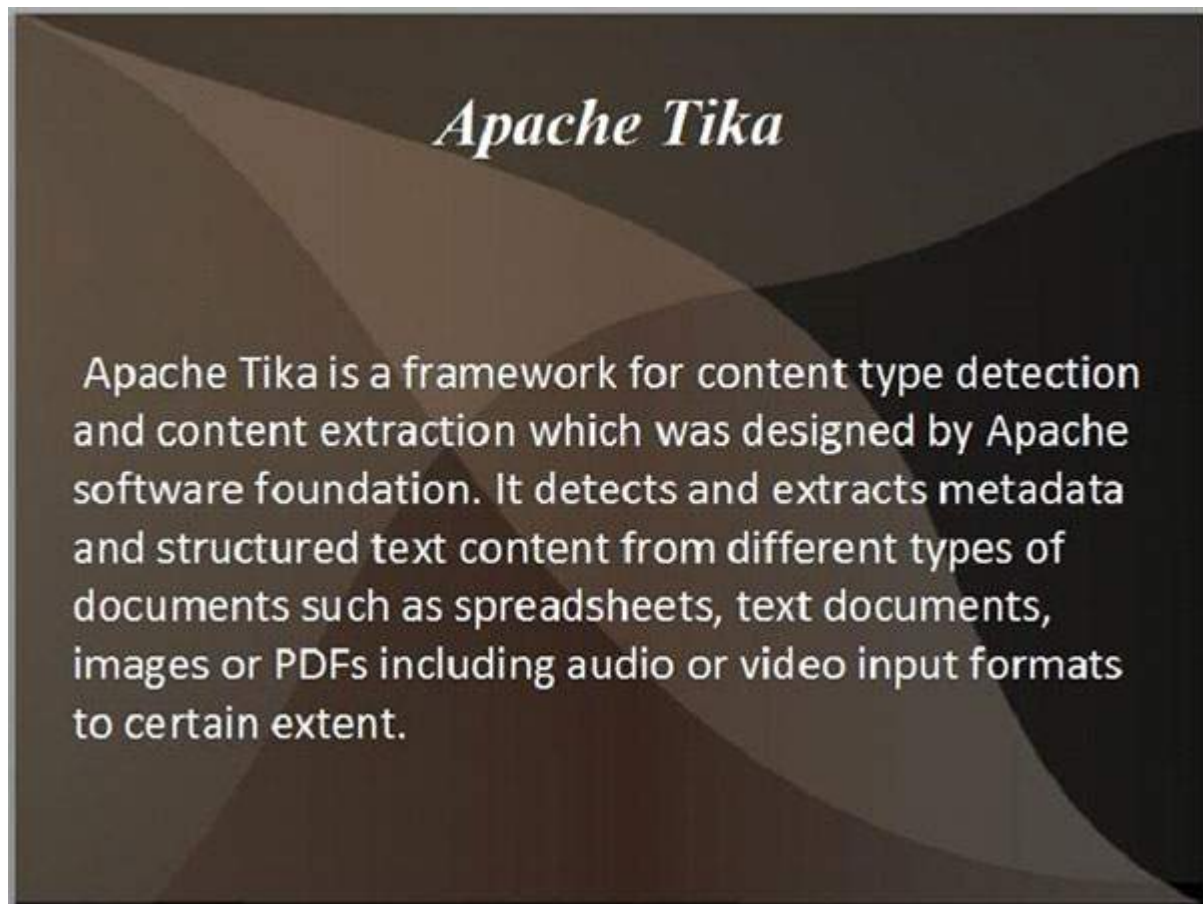
```
String[] metadataNames = metadata.names();

for(String name : metadataNames) {
    System.out.println(name + ": " + metadata.get(name));
}
}
```

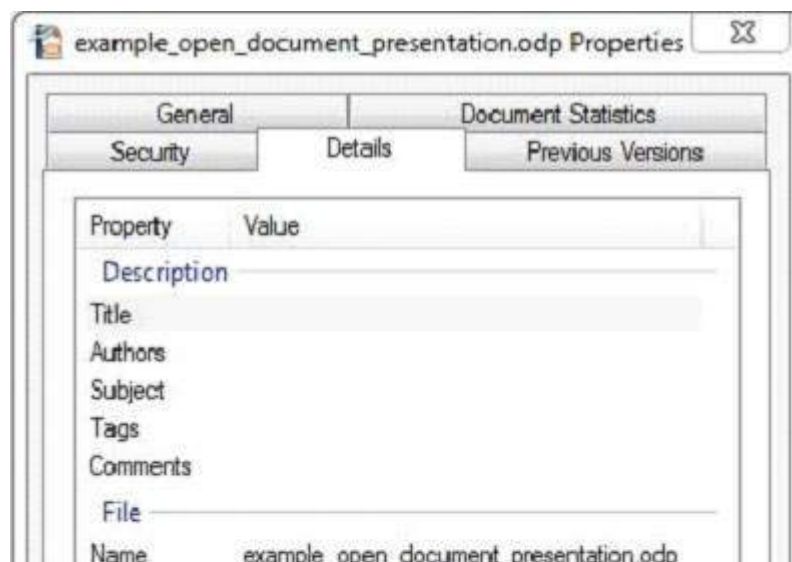
Save the above code as **OpenDocumentParse.java**, and compile it in the command prompt by using the following commands:

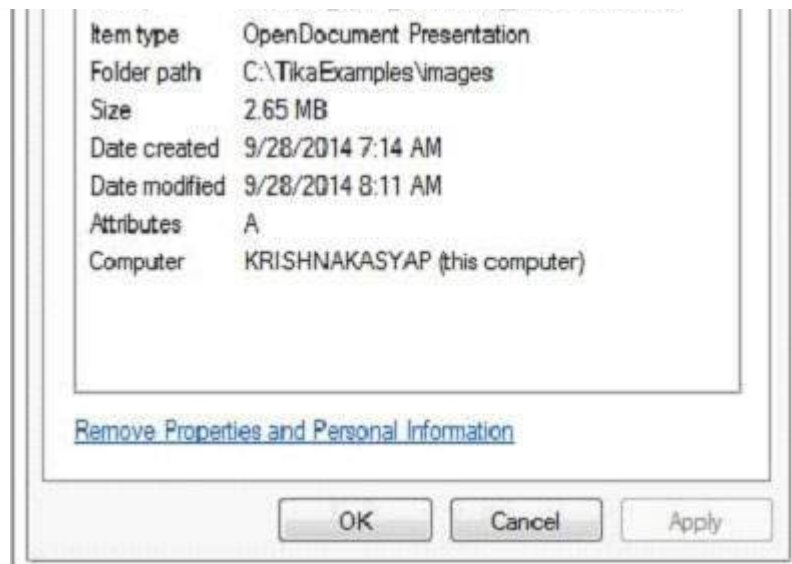
```
javac OpenDocumentParse.java
java OpenDocumentParse
```

The application will ask you to enter the path of the ODF that is to be parsed. Here we are passing the following sample open document presentation:



This document has the following properties:





After compiling the program, enter the full path of the ODF that is to be extracted.

```
enter path of your odf file:  
C:\TikaExamples\example_open_document_presentation.odp
```

Output:

Contents of the document:

Apache Tika

Apache Tika is a framework for content type detection and content extraction which was designed by Apache software foundation. It detects and extracts metadata and structured text content from different types of documents such as spreadsheets, text documents, images or PDFs including audio or video input formats to certain extent.

Metadata of the document:

```
editing-cycles: 4  
meta:creation-date: 2009-04-16T11:32:32.86  
dcterms:modified: 2014-09-28T07:46:13.03  
meta:save-date: 2014-09-28T07:46:13.03  
Last-Modified: 2014-09-28T07:46:13.03  
dcterms:created: 2009-04-16T11:32:32.86  
date: 2014-09-28T07:46:13.03  
modified: 2014-09-28T07:46:13.03  
nbObject: 36  
Edit-Time: PT32M6S  
Creation-Date: 2009-04-16T11:32:32.86  
Object-Count: 36  
meta:object-count: 36  
generator: OpenOffice/4.1.0$Win32 OpenOffice.org_project/410m18$Build-9764  
Content-Type: application/vnd.oasis.opendocument.presentation  
Last-Save-Date: 2014-09-28T07:46:13.03
```

TIKA - EXTRACTING MS OFFICE FILES

Given below is the program to extract content and metadata from a Microsoft Office Document.

```
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.Scanner;  
  
import org.apache.tika.exception.TikaException;  
import org.apache.tika.metadata.Metadata;  
import org.apache.tika.parser.ParseContext;  
import org.apache.tika.parser.microsoft.ooxml.OOXMLParser;
```

```

import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class MSeExcelParse {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the path of your MS-Office Document:");
        String fileName = scanner.nextLine();

        //detecting the file type
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputstream = new FileInputStream(new File(fileName));
        ParseContext pcontext = new ParseContext();

        //OOXML parser
        OOXMLParser msofficeparser = new OOXMLParser();
        msofficeparser.parse(inputstream, handler, metadata, pcontext);
        System.out.println("Contents of the document:" + handler.toString());
        System.out.println("Metadata of the document:");
        String[] metadataNames = metadata.names();

        for(String name : metadataNames) {
            System.out.println(name + ": " + metadata.get(name));
        }
    }
}

```

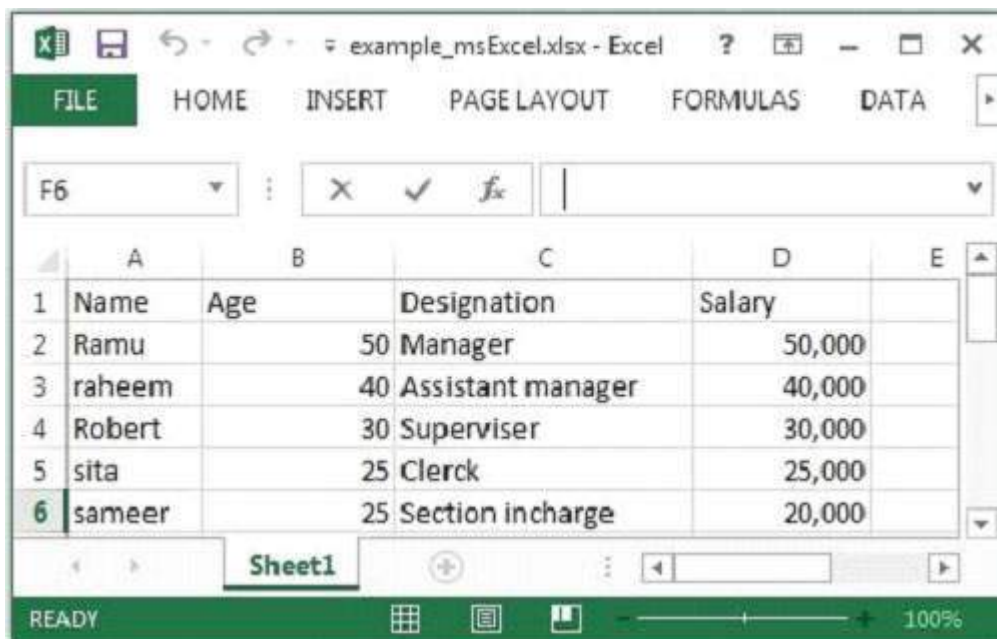
Save the above code as **MSeExcelParse.java**, and compile it from the command prompt by using the following commands:

```

javac MSeExcelParse.java
java MSeExcelParse

```

Here we are passing the following sample Excel file

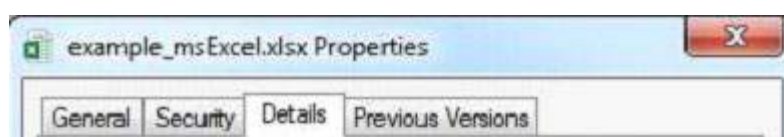


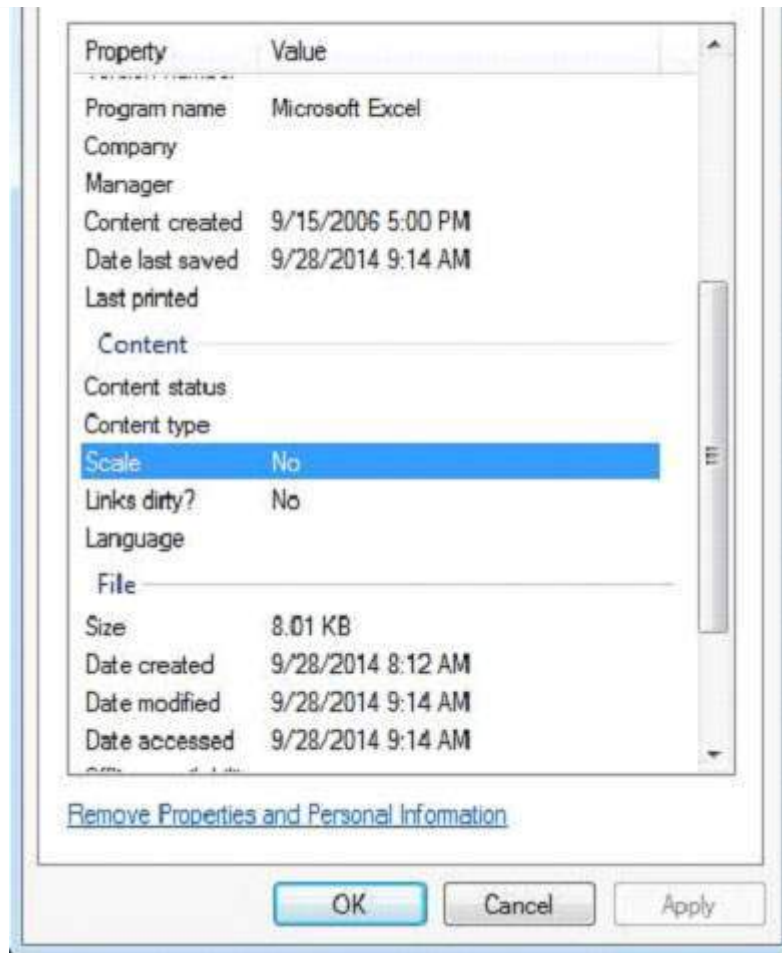
The screenshot shows an Excel window titled 'example_msExcel.xlsx - Excel'. The 'FILE' tab is selected. The active cell is F6. The table data is as follows:

	A	B	C	D	E
1	Name	Age	Designation	Salary	
2	Ramu	50	Manager	50,000	
3	raheem	40	Assistant manager	40,000	
4	Robert	30	Supervisor	30,000	
5	sita	25	Clerck	25,000	
6	sameer	25	Section incharge	20,000	

The status bar at the bottom shows 'READY', 'Sheet1', and a zoom level of 100%.

The given Excel file has the following properties:





The application will ask you to enter the path of the MS-Office document that is to be parsed.

Enter the path of your MS-Office Document:
C:\TikaExamples\example_msExcel.xlsx

Output:

Contents of the document:

Sheet1

Name Age Designation Salary

Ramu 50 Manager 50,000

Raheem 40 Assistant manager 40,000

Robert 30 Supervisor 30,000

sita 25 Clerk 25,000

sameer 25 Section in-charge 20,000

Metadata of the document:

meta:creation-date: 2006-09-16T00:00:00Z

dcterms:modified: 2014-09-28T15:18:41Z

meta:save-date: 2014-09-28T15:18:41Z

Application-Name: Microsoft Excel

extended-properties:Company:

dcterms:created: 2006-09-16T00:00:00Z

Last-Modified: 2014-09-28T15:18:41Z

Application-Version: 15.0300

date: 2014-09-28T15:18:41Z

publisher:

modified: 2014-09-28T15:18:41Z

Creation-Date: 2006-09-16T00:00:00Z

extended-properties:AppVersion: 15.0300

protected: false

dc:publisher:

extended-properties:Application: Microsoft Excel

Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet

Last-Save-Date: 2014-09-28T15:18:41Z

TIKA - EXTRACTING TEXT DOCUMENT

Given below is the program to extract content and metadata from a Text document:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Scanner;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.sax.BodyContentHandler;
import org.apache.tika.parser.txt.TXTParser;

import org.xml.sax.SAXException;

public class TextParser {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the path of your file:");
        String fileName = scanner.nextLine();

        //detecting the file type
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputStream = new FileInputStream(new File(fileName));
        ParseContext pcontext = new ParseContext();

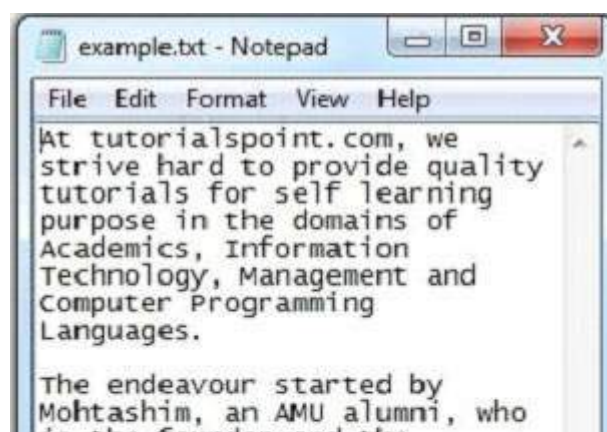
        //Text document parser
        TXTParser TextParser = new TXTParser();
        TextParser.parse(inputStream, handler, metadata, pcontext);
        System.out.println("Contents of the document:" + handler.toString());
        System.out.println("Metadata of the document:");
        String[] metadataNames = metadata.names();

        for(String name : metadataNames) {
            System.out.println(name + ": " + metadata.get(name));
        }
    }
}
```

Save the above code as **TextParser.java**, and compile it from the command prompt by using the following commands:

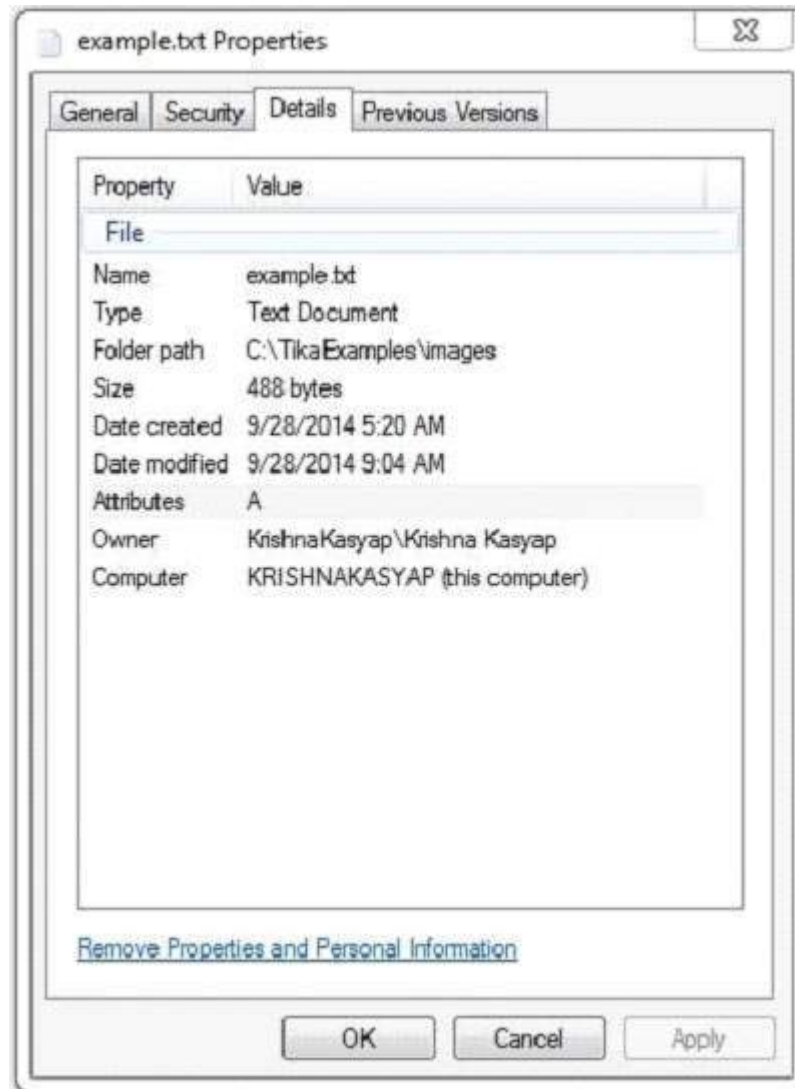
```
javac TextParser .java
java TextParser
```

Here we are passing the following sample text document:



is the founder and the managing director of Tutorials Point (I) Pvt. Ltd. He came up with the website tutorialspoint.com in year 2006 with the help of handpicked freelancers, with an array of tutorials for computer programming languages.

The text document has the following properties:



The application will ask you to enter the path of the text document that is to be parsed.

Enter the path of your file:
C:\TikaExamples\example.txt

Output:

Contents of the document:

At tutorialspoint.com, we strive hard to provide quality tutorials for self-learning purpose in the domains of Academics, Information Technology, Management and Computer Programming Languages.

The endeavour started by Mohtashim, an AMU alumni, who is the founder and the managing director of Tutorials Point (I) Pvt. Ltd. He came up with the website tutorialspoint.com in year 2006 with the help of handpicked freelancers, with an array of tutorials for

computer programming languages.

Metadata of the document:

Content-Encoding: windows-1252

Content-Type: text/plain; charset=windows-1252

TIKA - EXTRACTING HTML DOCUMENT

Given below is the program to extract content and metadata from an HTML document.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Scanner;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.html.HtmlParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class HtmlParse {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the path of your file");
        String fileName = scanner.nextLine();

        //detecting the file type
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputstream = new FileInputStream(new File(fileName));
        ParseContext pcontext = new ParseContext();

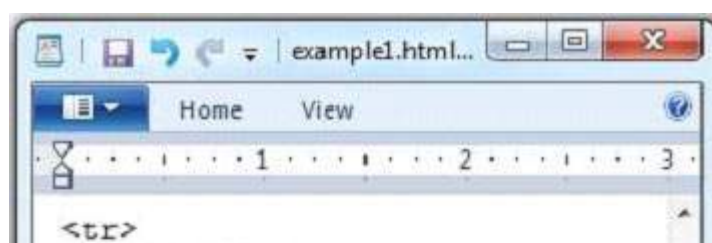
        //Html parser
        HtmlParser htmlparser = new HtmlParser();
        htmlparser.parse(inputstream, handler, metadata, pcontext);
        System.out.println("Contents of the document:" + handler.toString());
        System.out.println("Metadata of the document:");
        String[] metadataNames = metadata.names();

        for(String name : metadataNames) {
            System.out.println(name + ": " + metadata.get(name));
        }
    }
}
```

Save the above code as **HtmlParse.java**, and compile it from the command prompt by using the following commands:

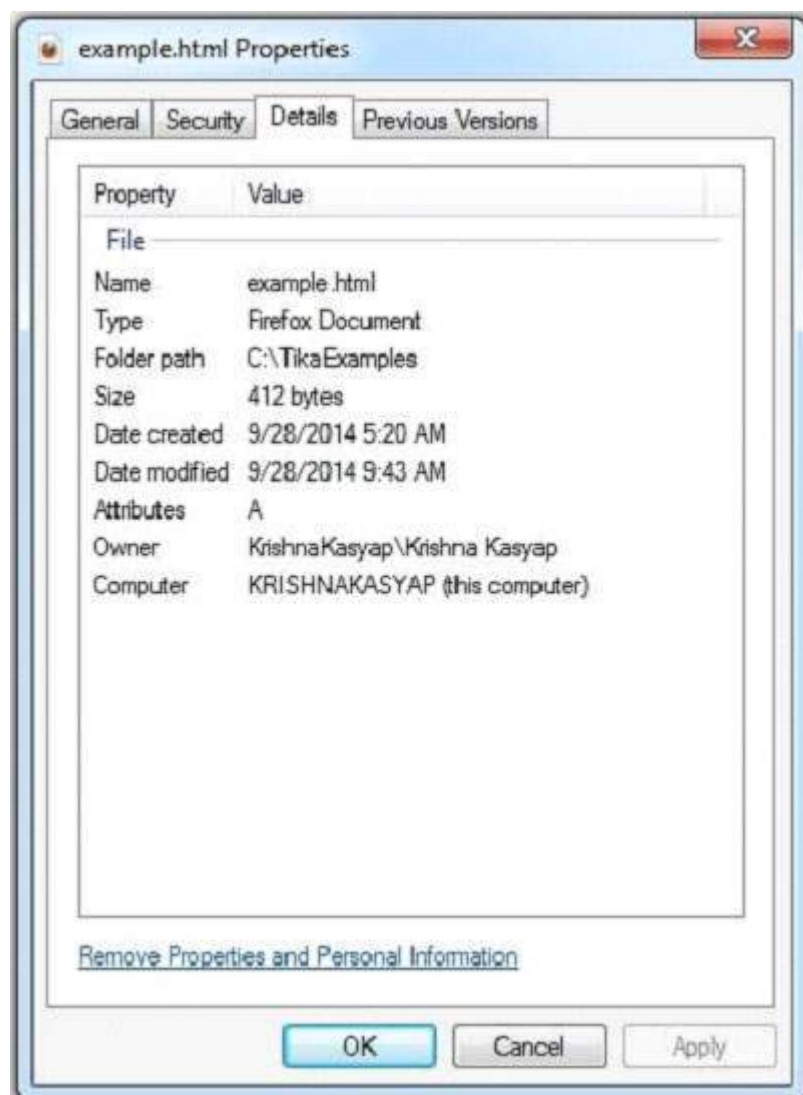
```
javac HtmlParse.java
java HtmlParse
```

Here we are passing the following sample HTML document.



```
<th>Name</th>
<th>Salary</th>
<th>age</th>
</tr>
<tr>
<td>Ramesh Raman</td>
<td>50000</td>
<th>20</th>
</tr>
<tr>
<td>Shabbir Hussein</td>
<td>70000</td>
<th>25</th>
</tr>
<tr>
<td>umesh Raman</td>
<td>50000</td>
<th>30</th>
</tr>
</table>
</body>
</html>
```

The HTML document has the following properties:



The application will ask you to enter the path of the HTML document that is to be parsed.

```
Enter the path of your file:  
C:\TikaExamples\example1.html
```

Output:

```
Contents of the document:  
Name           Salary      age  
Ramesh Raman    50000      20  
Shabbir Hussein 70000      25  
Umesh Raman     50000      30  
Somesh          50000      35  
Metadata of the document:  
title:  HTML Table Header  
Content-Encoding:  windows-1252  
Content-Type:  text/html; charset=windows-1252  
dc:title:  HTML Table Header
```

TIKA - EXTRACTING XML DOCUMENT

Given below is the program to extract content and metadata from an XML document:

```
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.Scanner;  
  
import org.apache.tika.exception.TikaException;  
import org.apache.tika.metadata.Metadata;  
import org.apache.tika.parser.ParseContext;  
import org.apache.tika.parser.xml.XMLParser;  
import org.apache.tika.sax.BodyContentHandler;  
  
import org.xml.sax.SAXException;  
  
public class XmlParse {  
  
    public static void main(final String[] args) throws IOException, SAXException,  
TikaException {  
  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Enter the path of your file:");  
        String fileName = scanner.nextLine();  
  
        //detecting the file type  
        BodyContentHandler handler = new BodyContentHandler();  
        Metadata metadata = new Metadata();  
        FileInputStream inputStream = new FileInputStream(new File(fileName));  
        ParseContext pcontext = new ParseContext();  
  
        //Xml parser  
        XMLParser xmlparser = new XMLParser();  
        xmlparser.parse(inputStream, handler, metadata, pcontext);  
        System.out.println("Contents of the document:" + handler.toString());  
        System.out.println("Metadata of the document:");  
        String[] metadataNames = metadata.names();  
  
        for(String name : metadataNames) {  
            System.out.println(name + ": " + metadata.get(name));  
        }  
    }  
}
```

Save the above code as **XmlParse.java**, and compile it from the command prompt by using the following commands:

```
javac XmlParse.java
java XmlParse
```

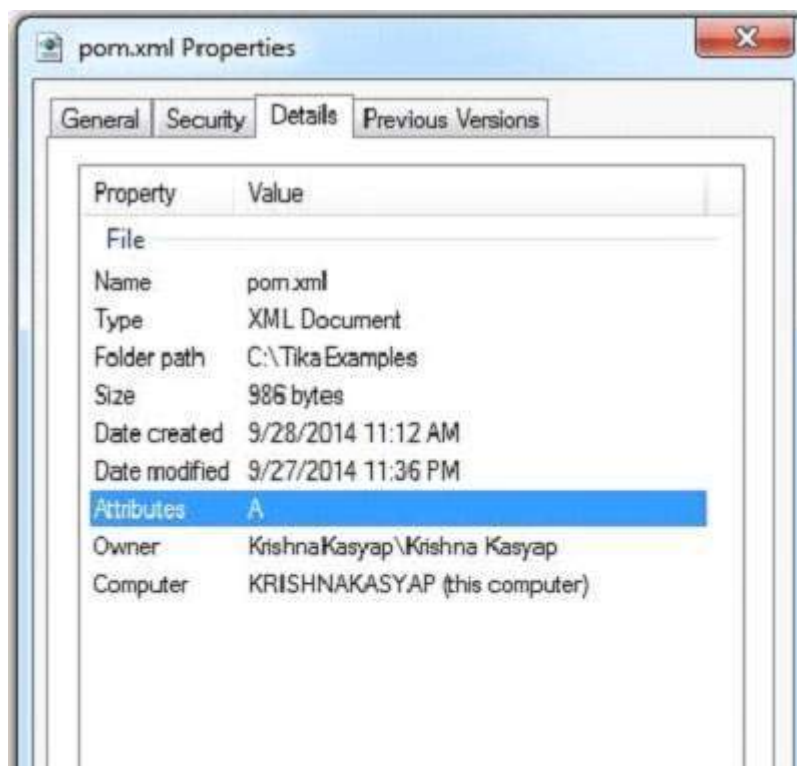
Here we are passing the following sample XML document.



The screenshot shows a Notepad window titled "pom.xml - Notepad". The window contains the following XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<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>org.apache.tika</groupId>
  <artifactId>tika</artifactId>
  <version>1.6</version>
  <dependencies>
    <dependency>
      <groupId>org.apache.tika</groupId>
      <artifactId>tika-core</artifactId>
      <version>1.6</version>
    </dependency>
    <dependency>
      <groupId>org.apache.tika</groupId>
      <artifactId>tika-parsers</artifactId>
      <version>1.6</version>
    </dependency>
  </dependencies>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.1</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

This document has the following properties:



The screenshot shows the "pom.xml Properties" dialog box with the "Details" tab selected. The dialog displays the following properties:

Property	Value
File	
Name	pom.xml
Type	XML Document
Folder path	C:\TikaExamples
Size	986 bytes
Date created	9/28/2014 11:12 AM
Date modified	9/27/2014 11:36 PM
Attributes	A
Owner	KrishnaKasyap\Krishna Kasyap
Computer	KRISHNAKASYAP (this computer)



The application will ask you to enter the path of XML document that is to be parsed.

```
Enter the path of your file:  
C:\TikaExamples\pom.xml
```

Output:

Contents of the document:

```
4.0.0  
org.apache.tika  
tika  
1.6  
org.apache.tika  
tika-core  
1.6  
org.apache.tika  
tika-parsers  
1.6  
src  
maven-compiler-plugin  
3.1  
1.7  
1.7
```

Metadata of the document:

```
Content-Type: application/xml
```

TIKA - EXTRACTING .CLASS FILE

Given below is the program to extract content and metadata from a .class file.

```
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.Scanner;  
  
import org.apache.tika.exception.TikaException;  
import org.apache.tika.metadata.Metadata;  
import org.apache.tika.parser.ParseContext;  
import org.apache.tika.parser.asm.ClassParser;  
import org.apache.tika.sax.BodyContentHandler;  
  
import org.xml.sax.SAXException;  
  
public class JavaClassParse {  
  
    public static void main(final String[] args) throws IOException, SAXException,  
TikaException {  
  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Enter the path of your file:");  
        String fileName = scanner.nextLine();  
  
        //detecting the file type  
        BodyContentHandler handler = new BodyContentHandler();  
        Metadata metadata = new Metadata();
```

```

FileInputStream inputStream = new FileInputStream(new File(fileName));
ParseContext pcontext = new ParseContext();

//Html parser
ClassParser ClassParser =new ClassParser();
ClassParser.parse(inputStream, handler, metadata,pcontext);
System.out.println("Contents of the document:" + handler.toString());
System.out.println("Metadata of the document:");
String[] metadataNames = metadata.names();

for(String name : metadataNames) {
    System.out.println(name + ":    " + metadata.get(name));
}
}
}

```

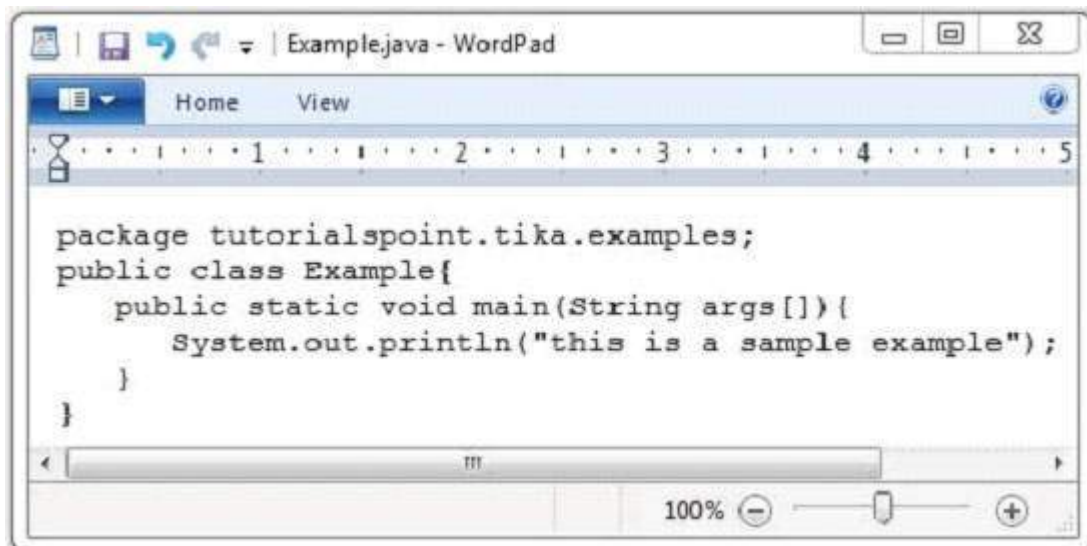
Save the above code as **JavaClassParse.java**, and compile it from the command prompt by using the following commands:

```

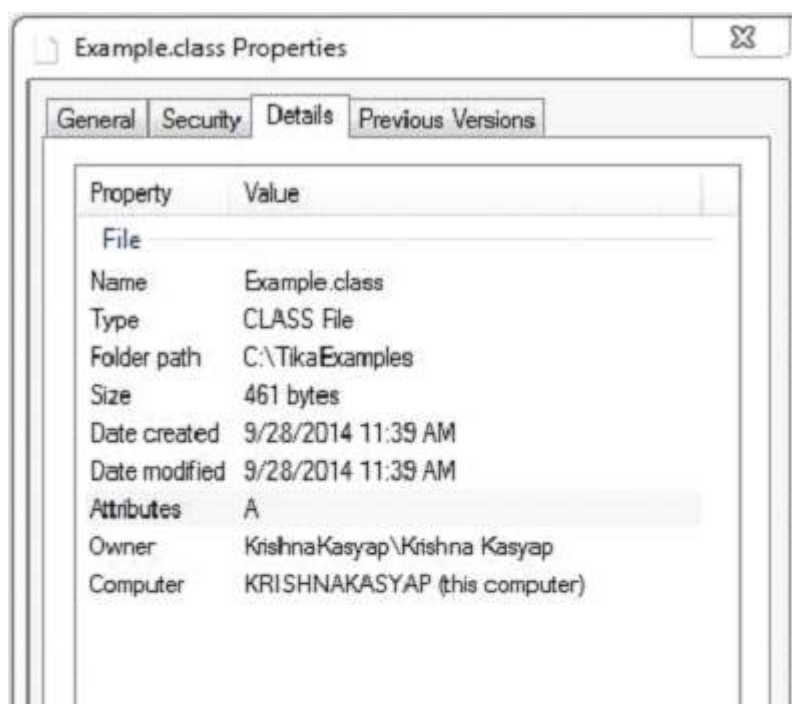
javac JavaClassParse.java
java JavaClassParse

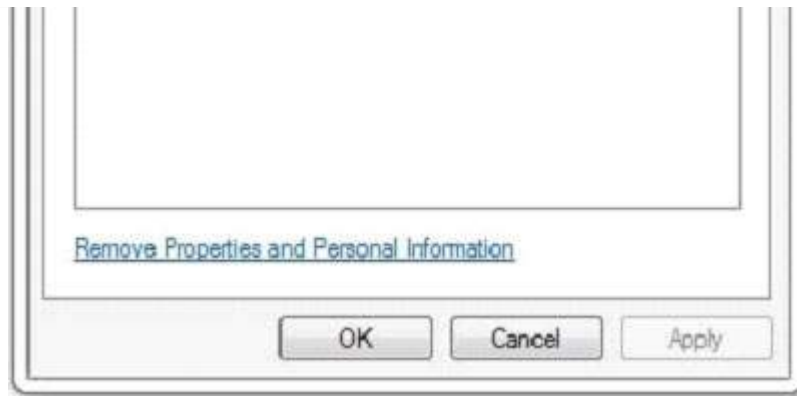
```

Here we are passing the .class file of the following **Example.java** file:



And this .class file has the following properties:





The application will ask you to enter the path class file that is to be parsed.

```
Enter the path of your file:  
C:\TikaExamples\Example.class
```

Output:

```
Contents of the document:  
package tutorialspoint.tika.examples;  
public synchronized class Example {  
    public void Example();  
    public static void main(String[]);  
}
```

```
Metadata of the document:  
title: Example  
resourceName: Example.class  
dc:title: Example
```

TIKA - EXTRACTING JAR FILE

Given below is the program to extract content and metadata from a Java Archive *jar* file:

```
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.Scanner;  
  
import org.apache.tika.exception.TikaException;  
import org.apache.tika.metadata.Metadata;  
import org.apache.tika.parser.ParseContext;  
import org.apache.tika.sax.BodyContentHandler;  
import org.apache.tika.parser.pkg.PackageParser;  
  
import org.xml.sax.SAXException;  
  
public class PackageParse {  
  
    public static void main(final String[] args) throws IOException, SAXException,  
TikaException {  
  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Enter the path of your file:");  
        String fileName = scanner.nextLine();  
  
        //detecting the file type  
        BodyContentHandler handler = new BodyContentHandler();  
        Metadata metadata = new Metadata();  
        FileInputStream inputstream = new FileInputStream(new File(fileName));  
        ParseContext pcontext=new ParseContext();  
  
        //Package parser  
        PackageParser packageparser =new PackageParser();  
        packageparser.parse(inputstream, handler, metadata, pcontext);
```



```

System.out.println("Contents of the document: "+handler.toString());
System.out.println("Metadata of the document:");
String[] metadataNames = metadata.names();

for(String name : metadataNames) {
    System.out.println(name + ": " + metadata.get(name));
}
}
}

```

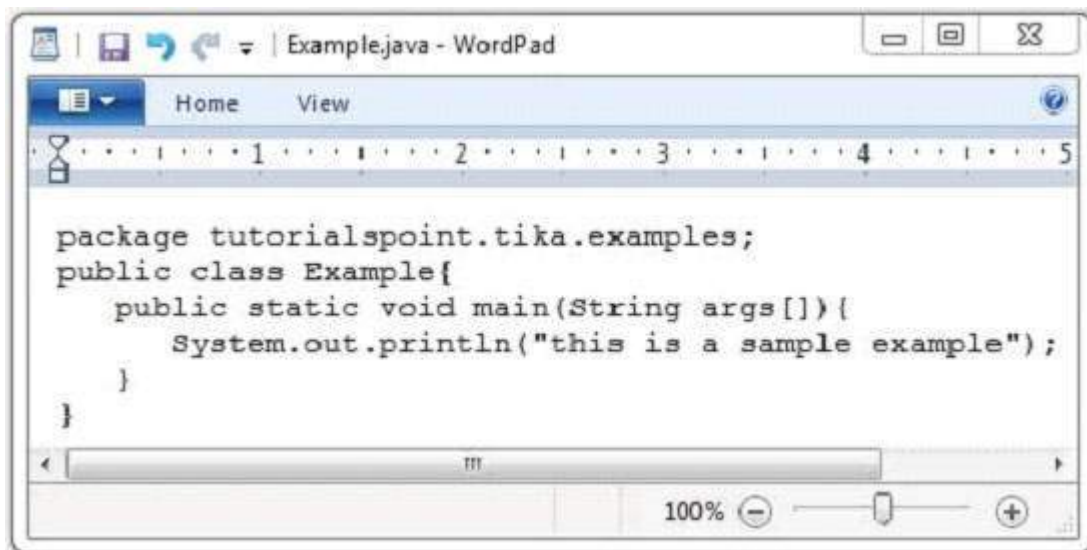
Save the above code as **PackageParse.java**, and compile it from the command prompt by using the following commands:

```

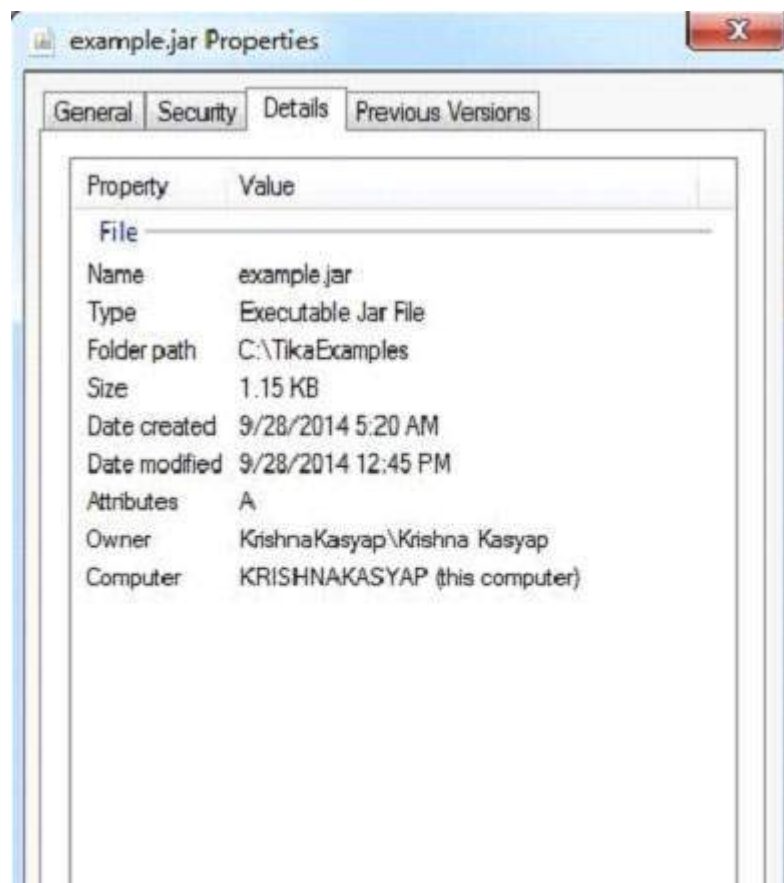
javac PackageParse.java
java PackageParse

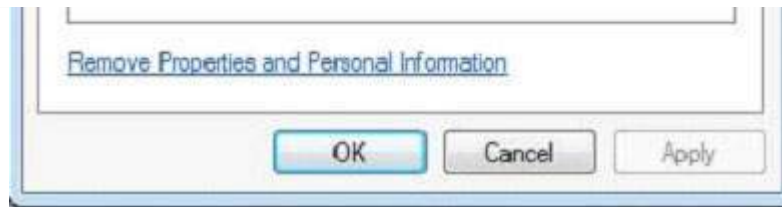
```

Here we are passing the jar file of the following Example.java file.



The jar file has the following properties:





The application will ask you to enter the path of the jar file that is to be parsed.

Enter the path of your file:
C:\TikaExamples\example.jar

Output:

Contents of the document:
META-INF/MANIFEST.MF
tutorialspoint/tika/examples/Example.class

Metadata of the document:
Content-Type: application/zip

TIKA - EXTRACTING IMAGE FILE

Given below is the program to extract content and metadata from a JPEG image.

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

import java.util.Scanner;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.jpeg.JpegParser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class JpegParse {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the path of your file:");
        String fileName = scanner.nextLine();

        //detecting the file type
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputstream = new FileInputStream(new File(fileName));
        ParseContext pcontext=new ParseContext();

        //Jpeg Parse
        JpegParser JpegParser = new JpegParser();
        JpegParser.parse(inputstream, handler, metadata, pcontext);
        System.out.println("Contents of the document:" +handler.toString());
        System.out.println("Metadata of the document:");
        String[] metadataNames = metadata.names();

        for(String name : metadataNames) {
            System.out.println(name + ": " + metadata.get(name));
        }
    }
}
```

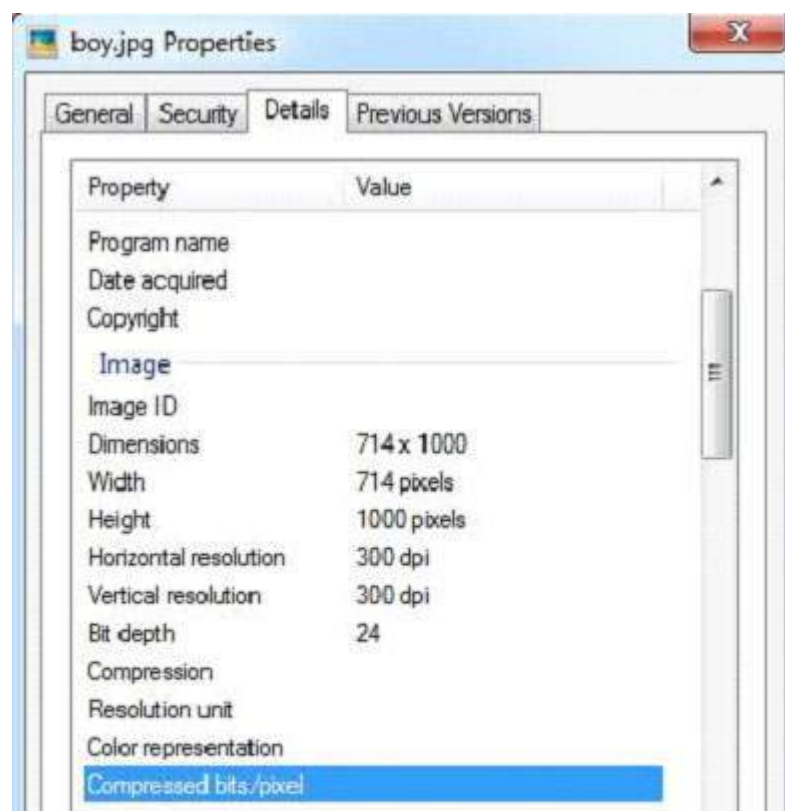
Save the above code as **JpegParse.java**, and compile it from the command prompt by using the following commands:

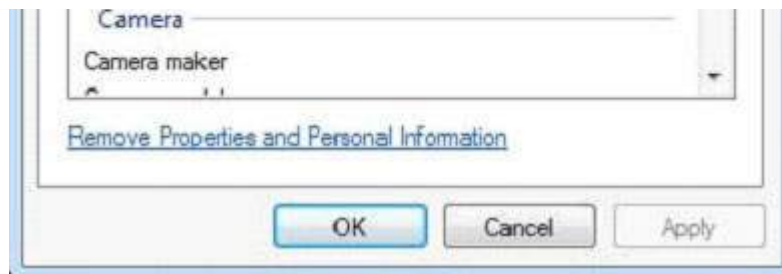
```
javac JpegParse.java
java JpegParse
```

Here we are passing the following JPEG file:



The JPEG file has the following properties:





The application will ask you to enter the path of the JPEG file that is to be parsed.

Enter the path of your file:
C:\TikaExamples\boy.jpg

Output:

Contents of the document:

Metadata of the document:

IPTC-NAA record: 92 bytes binary data

Number of Components: 3

Image Height: 1000 pixels

Resolution Units: inch

Data Precision: 8 bits

tiff:BitsPerSample: 8

Compression Type: Baseline

Component 1: Y component: Quantization table 0, Sampling factors 1 horiz/1 vert

Component 2: Cb component: Quantization table 1, Sampling factors 1 horiz/1 vert

tiff:ImageLength: 1000

Component 3: Cr component: Quantization table 1, Sampling factors 1 horiz/1 vert

X Resolution: 300 dots

tiff:ImageWidth: 714

Application Record Version: 4

Image Width: 714 pixels

Original Transmission Reference:

53616c7465645f5fd22a84941585d89cc735d889c9d5ac58a01faf2c92ee3c6f9bcb38359bbe1eef

Y Resolution: 300 dots

TIKA - EXTRACTING MP4 FILES

Given below is the program to extract content and metadata from mp4 files:

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

import java.util.Scanner;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.mp4.MP4Parser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class Mp4Parse {

    public static void main(final String[] args) throws IOException, SAXException,
TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the path of your file");
        String fileName = scanner.nextLine();

        //detecting the file type
```

```

BodyContentHandler handler = new BodyContentHandler();
Metadata metadata = new Metadata();
FileInputStream inputStream = new FileInputStream(new File(fileName));
ParseContext pcontext = new ParseContext();

//Html parser
MP4Parser MP4Parser = new MP4Parser();
MP4Parser.parse(inputStream, handler, metadata, pcontext);
System.out.println("Contents of the document: " + handler.toString());
System.out.println("Metadata of the document:");
String[] metadataNames = metadata.names();

for(String name : metadataNames) {
    System.out.println(name + ": " + metadata.get(name));
}
}
}

```

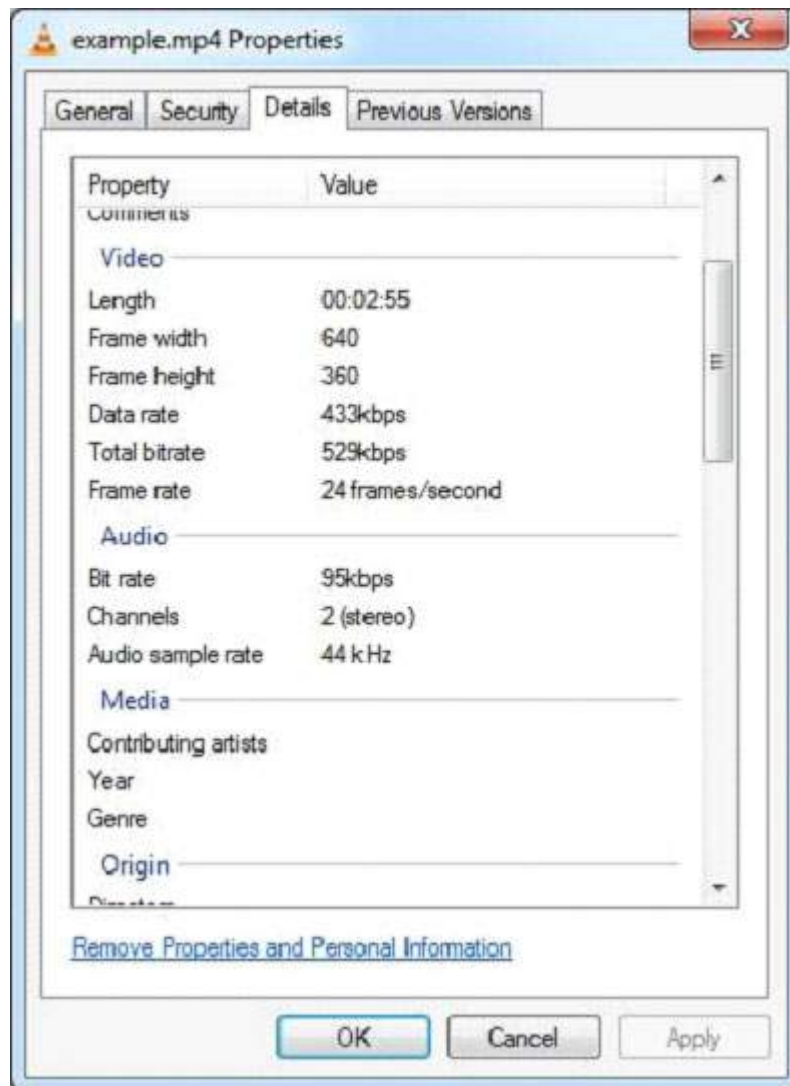
Save the above code as JpegParse.java, and compile it from the command prompt by using the following commands:

```

javac Mp4Parse.java
java Mp4Parse

```

Here we are passing an mp4 file having the following properties:



The application will ask you to enter the path of the mp4 file that is to be parsed.

```

Enter the path of your file:
C:\TikaExamples\example.mp4

```

Output:

Contents of the document:

Metadata of the document:

dcterms:modified: 2014-01-06T12:10:27Z
meta:creation-date: 1904-01-01T00:00:00Z
meta:save-date: 2014-01-06T12:10:27Z
Last-Modified: 2014-01-06T12:10:27Z
dcterms:created: 1904-01-01T00:00:00Z
date: 2014-01-06T12:10:27Z
tiff:ImageLength: 360
modified: 2014-01-06T12:10:27Z
Creation-Date: 1904-01-01T00:00:00Z
tiff:ImageWidth: 640
Content-Type: video/mp4
Last-Save-Date: 2014-01-06T12:10:27Z

TIKA - EXTRACTING MP3 FILES

Given below is the program to extract content and metadata from mp3 files:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Scanner;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.mp3.LyricsHandler;
import org.apache.tika.parser.mp3.Mp3Parser;
import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

public class Mp3Parse {
    public static void main(final String[] args) throws Exception, IOException,
        SAXException, TikaException {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the path of your file:");
        String fileName = scanner.nextLine();
        System.out.println(fileName);

        //detecting the file type
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream inputstream = new FileInputStream(new File(fileName));
        ParseContext pcontext = new ParseContext();

        //Mp3 parser
        Mp3Parser Mp3Parser = new Mp3Parser();
        Mp3Parser.parse(inputstream, handler, metadata, pcontext);
        LyricsHandler lyrics = new LyricsHandler(inputstream, handler);

        while(lyrics.hasLyrics()) {
            System.out.println(lyrics.toString());
        }

        System.out.println("Contents of the document:" + handler.toString());
        System.out.println("Metadata of the document:");
        String[] metadataNames = metadata.names();

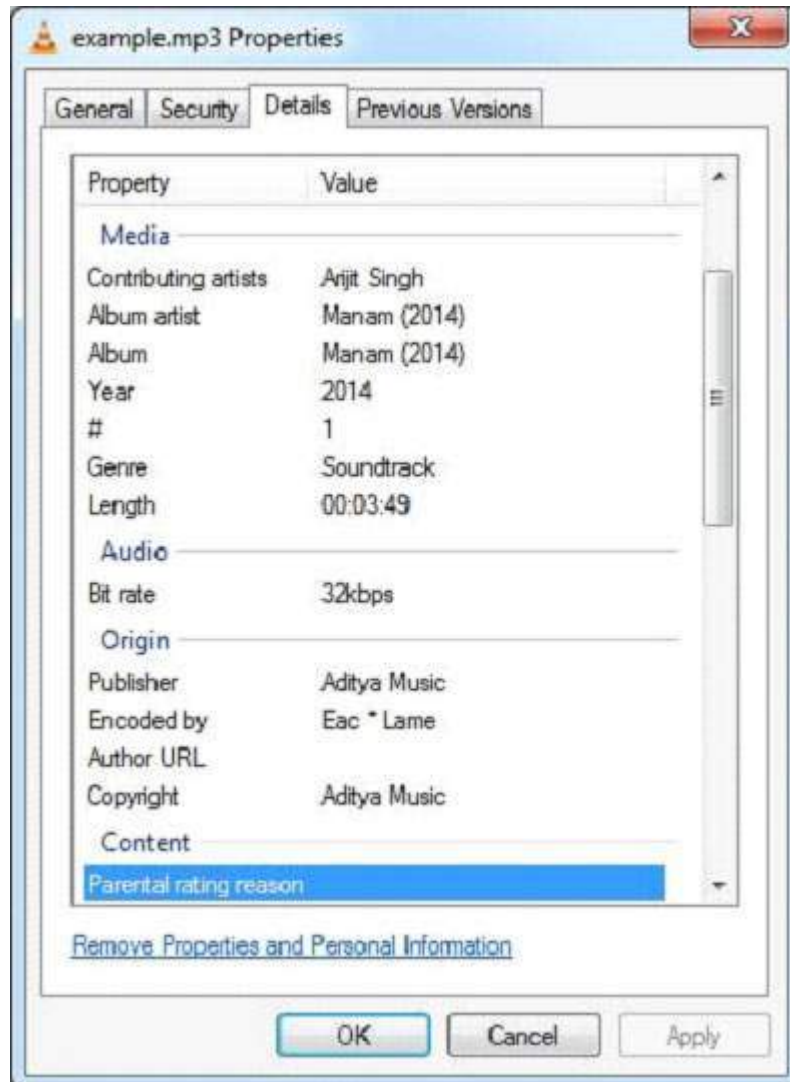
        for(String name : metadataNames) {
            System.out.println(name + ": " + metadata.get(name));
        }
    }
}
```

```
}
```

Save the above code as **JpegParse.java**, and compile it from the command prompt by using the following commands:

```
javac Mp3Parse.java  
java Mp3Parse
```

Here we are passing an mp3 file having the following properties:



The application will ask you to enter the path of the mp3 file that is to be parsed.

```
Enter the path of your file:  
C:\TikaExamples\example.mp3
```

If the given file has any lyrics, our application will capture and display that along with the output.

Output:

```
Contents of the document:  
Kanulanu Thaake  
Arijit Singh  
Manam (2014), track 01/06  
2014  
Soundtrack  
30171.65  
eng -  
DRGM  
Arijit Singh  
Manam (2014), track 01/06
```

2014
Soundtrack
30171.65
eng -
DRGM

Metadata of the document:

xmpDM:releaseDate: 2014
xmpDM:duration: 30171.650390625
xmpDM:audioChannelType: Stereo
dc:creator: Arijit Singh
xmpDM:album: Manam (2014)
Author: Arijit Singh
xmpDM:artist: Arijit Singh
channels: 2
xmpDM:audioSampleRate: 44100
xmpDM:logComment: eng -
DRGM
xmpDM:trackNumber: 01/06
version: MPEG 3 Layer III Version 1
creator: Arijit Singh
xmpDM:composer: Music : Anoop Rubens | Lyrics : Vanamali
xmpDM:audioCompressor: MP3
title: Kanulanu Thaake
samplerate: 44100
meta:author: Arijit Singh
xmpDM:genre: Soundtrack
Content-Type: audio/mpeg
xmpDM:albumArtist: Manam (2014)
dc:title: Kanulanu Thaake

Processing math: 100%