```
package org.codehaus.bayesian;

import javax.mail.internet.*;
import java.io.*;
import java.util.*;
import java.util.regex.*;

/**
 This class is responsible for extracting all the useful information from
 E-Mails. It parses MIME E-Mail messages into Strings.
 */
public class Parser
{
    /**
     Parse an entire message. This will be called initially by the Filter
     It will call other methods within this class to fully parse the message and
     return the string.
     This class is responsible from converting a message from a MIME type to a
     string containing all the relevant text from the message.

     @param m the e-mail you want parsed.
     @return String the relevant words from the e-mail that will be considered in
the
     calculation of the probability that the e-mail is spam.
     */
    public String parse( Message m )
    {
        // parse the header
        String parsedHeader = parseHeader( m.getHeader() );

        String x = m.getContentType();
        if ( x != null )
        {

            if ( x.startsWith( "t" ) || x.startsWith( "T" ) )
            {
                // if content type is text/plain then just get the body and  return
it.
                if ( x.equalsIgnoreCase( "text/plain" ) )
                {
                    String parsedBody = parseTextPlain( m.getBody(),
m.getContentTransferEncoding() );
                    String totalParsed = parsedHeader + " " + parsedBody;
                    return totalParsed;
                }
                // if content type is text/html parse out the html tags then return.
                else if ( x.equalsIgnoreCase( "text/html" ) )
                {
                    String parsedBody = parseTextHtml( m.getBody(),
m.getContentTransferEncoding() );
                    String totalParsed = parsedHeader + " " + parsedBody;
                    return totalParsed;

                }
                // if content type is text/enriched use html parse too.
                else if ( x.equalsIgnoreCase( "text/enriched" ) )
                {
                    String parsedBody = parseTextPlain( m.getBody(),
m.getContentTransferEncoding() );
                    String totalParsed = parsedHeader + " " + parsedBody;
                    return totalParsed;

                }
            }
            else if ( x.startsWith( "mu" ) || x.startsWith( "MU" ) )
            {
                /*
                    Parse the Multipart give parameters
                        the body: what the parse is going to work with.
                        the content type: for further analysis.
                        the boundary: for breaking up the body.
                */
                String parsedBody = parseMultipart( m.getBody(), m.getHeader(), x,
m.getBoundary() );
                String totalParsed = parsedHeader + " " + parsedBody;
                return totalParsed;
            }
            else if ( x.startsWith( "me" ) || x.startsWith( "ME" ) )
            {
                // if content type is message/rfc822 extract message and parse it
                if ( x.equalsIgnoreCase( "message/rfc822" ) )
                {
                    String parsedBody = parseMessageRfc822( m.getBody(),
m.getContentTransferEncoding() );
                    return parsedHeader + " " + parsedBody;
                }
            }
        }
        return " ";
    }

    /**
     Parse the E-Mail header of the MIME message for information.

     @param header An ArrayList containing Strings which contain the header
information.

     @return String the parsed header information.
     */
    protected String parseHeader( ArrayList header )
    {
        StringBuffer buff = new StringBuffer();

        for ( int i = 0; i < header.size(); i++ )
        {
            String temp = (String) header.get( i );
            if ( temp.startsWith( "Received:" ) )
            {
                // extract any ip's from it and chop off the host.
                String ipReg = "(([0-9]{1,3}\\.){1,3}){3}([0-9]{1,3}){1}";
                try
                {
                    Pattern p = Pattern.compile( ipReg, Pattern.CASE_INSENSITIVE );
                    Matcher m = p.matcher( temp );
                    while ( m.find() )
                    {
                        int start = m.start();
                        int end = m.end();
                        String match = temp.substring( start, end );
                        end = match.lastIndexOf( "." );
                        match = match.substring( 0, end );
                        String token = "IP:" + match;
                        buff.append( token + " " );
                    }
                }
                catch ( Exception e )
                {
                    //squelch the error.
                }
            }
            else if ( temp.startsWith( "From:" ) )
            {
                // remove anything we dont need
                String reg = "[\",\\<\\>\\(\\)\\;]";
                temp = temp.replaceAll( reg, "" );

                // break it up into words
                StringTokenizer st = new StringTokenizer( temp );
                // get rid of first token From:
                st.nextToken();

                while ( st.hasMoreTokens() )
                {
                    String a = st.nextToken();
                    String token = "From:" + a;
                    buff.append( token + " " );
                }

            }
            else if ( temp.startsWith( "Subject:" ) )
            {
                // take First token
                // append each token to the first one and add to the buffer.

                // break it up into words
                StringTokenizer st = new StringTokenizer( temp );
                // get rid of first token From:
                st.nextToken();

                while ( st.hasMoreTokens() )
                {
                    String a = st.nextToken();
                    String token = "Subject:" + a;
                    buff.append( token + " " );
                }
            }

        }
        String temp = buff.toString();
        return temp;
    }

    /**
     Parse text/plain MIME messages or body parts.
     This method will extract all the information from the E-Mail message or body
part that has
     the MIME type text/plain. If it happens to be encoded it uses the JavaMail
classes to decode it.

     @param body An ArrayList of strings comprising the body of the Message or Body
part.
     @param cte the message or body part's Content-Transfer-Encoding (e.g. Base64)
     @return the parsed information contained in a String.
     */
    protected String parseTextPlain( ArrayList body, String cte )
    {

        StringBuffer buff = new StringBuffer();
        String temp = "";

        if ( cte != null )
        {
            try
            {
                StringBuffer tempbuff = new StringBuffer();

                for ( int i = 0; i < body.size(); i++ )
                {
                    tempbuff.append( body.get( i ) + " " );
                }

                String tempbuffString = tempbuff.toString();

                byte[] encodedData = tempbuffString.getBytes();
                ByteArrayInputStream input = new ByteArrayInputStream( encodedData
);
                InputStreamReader bb = new InputStreamReader( MimeUtility.decode(
input, cte ) );
                BufferedReader rr = new BufferedReader( bb );
                String testline = rr.readLine();
                while ( testline != null )
                {
                    buff.append( testline );
                    testline = rr.readLine();
                }

                temp = buff.toString();
            }
            catch ( Exception e )
            {
                e.printStackTrace();
            }
        }
        else
        {
            for ( int i = 0; i < body.size(); i++ )
            {
                buff.append( body.get( i ) + " " );
            }
            temp = buff.toString();
        }

        // a new buffer to hold the parsed data
        StringBuffer parsedBuffer = new StringBuffer();

        // these mark the head and tail position of the substring I want placed into
the parsed buffer
        // the string is from the end of the last match to the start of the new
match.
        // Had to do it this way (replaceAll was not working!)
        int head = 0;
        int tail = 0;

        // the regular expressions to strip out all the non essential tokens and the
other to find
        // all URLs in the text.
        String regexp = "(\".*\")|(BEGIN[.*]*END)|(>)|(<)|(_)|(-)|(\\*)|(#)|(<.*>)
([,\\.\\=\t\\*\\+\\\\])|(\\&nbsp\\;)|(\\<.*)|(href)";
        String htmlReg = "(http:[^\\s]*\\s)|(www.[^\\s]+\\s)";


        try
        {
            Pattern p = Pattern.compile( htmlReg, Pattern.CASE_INSENSITIVE );
            Matcher m = p.matcher( temp );
            while ( m.find() )
            {
                int start = m.start();
                int end = m.end();

                tail = start;
                String nonLink = temp.substring( head, tail );
                nonLink = nonLink.replaceAll( regexp, "" );
                parsedBuffer.append( nonLink + " " );
                head = end;

                String match = temp.substring( start, end );
                String fineTune = "[\\>\"\\<,]";
                match = match.replaceAll( fineTune, "" );

                parsedBuffer.append( getUrl( match ) + " " );
            }
        }
        catch ( Exception e )
        {
        }
        return temp;
    }

    /**
     Parse text html body parts. E-mail can be sent in html format
     (although most people hate it.) A common way to send a message is in
     multipart/alternative. This is where the message is sent in plain text
     and in html formal.
     This method will extract all the information from the html body part.

     @param body the bodypart that is in html format.
     @param cte the content-transfer-encoding for that part.
     @return the String containing all the relevant words from this bodypart
     */
    protected String parseTextHtml( ArrayList body, String cte )
    {

        StringBuffer buff = new StringBuffer();
        String temp = "";
        if ( cte != null )
        {
            try
            {
                StringBuffer tempbuff = new StringBuffer();

                for ( int i = 0; i < body.size(); i++ )
                {
                    tempbuff.append( body.get( i ) + " " );
                }

                String tempbuffString = tempbuff.toString();

                byte[] encodedData = tempbuffString.getBytes();
                ByteArrayInputStream input = new ByteArrayInputStream( encodedData
);
                InputStreamReader bb = new InputStreamReader( MimeUtility.decode(
input, cte ) );
                BufferedReader rr = new BufferedReader( bb );
                String testline = rr.readLine();
                while ( testline != null )
                {
                    buff.append( testline );
                    testline = rr.readLine();
                }

                temp = buff.toString();

            }
            catch ( Exception e )
            {
                e.printStackTrace();
            }
        }
        else
        {
            for ( int i = 0; i < body.size(); i++ )
            {
                buff.append( body.get( i ) + " " );
            }
            temp = buff.toString();
        }

        // take out the html comments <!-- -->
        String reg = "<![^>]*>";
        String reg2 = "[\\.,_+/\\-]"; //(http[^\\s]+\\s)
        String reg3 = "[\\<\"\\n\\t\\>=]";
        temp = temp.replaceAll( reg, "" );
        temp = temp.replaceAll( reg2, "" );
        temp = temp.replaceAll( reg3, " " );

        return temp;
    }

    /**
     Parse text enriched body parts.
     Enriched text is quite rare. It uses the same kind of syntax as
     html but with different tags. I'll reuse the html parser in order to
     parse enriched text.
     @param body the bodypart that's in enriched text format.
     @param cte The content-transfer-Encoding for this bodypart.
     @return the String containing all the relevant information from this
     bodypart.
     */
    protected String parseTextEnriched( ArrayList body, String cte )
    {
        return parseTextHtml( body, cte );
    }


    /**
     Parses a message body/bodypart that is of type multipart/*.

     All present and future subtypes of the "multipart" type must use an
     identical syntax.  Subtypes may differ in their semantics.
     Therefore we can reuse this code for all subtypes of multipart.

     @param b the body/bodypart we want to parse.
     @param h the header information.
     @param content_type the content-type for this body/bodypart.
     @param boundary the string identifying the MIME boundary.
     @return the string containing the relevant information from this body part and
     recursively any sub bodyparts.
     */
    protected String parseMultipart( ArrayList b, ArrayList h, String content_type,
String boundary )
    {
        ArrayList body = new ArrayList();
        body.addAll( h );
        body.addAll( b );
        /*
            Break Up the body into the relative parts.
        */

        // string that will contain all the info from the multipart message
        String parsed = "";

        // extract the body parts
        // make the end boundary
        String boundary2 = boundary + "--";

        // this is the list of body parts
        ArrayList bodyPartList = new ArrayList();
        // a string used as a temp storage
        String temp = "";

        // these will mark the start and end of body parts
        int start = -1;
        int end = -1;

        // Version 2 of algorithm 10/04/2003
        //
        // Step through the arraylist looking for the boundary. When we first find
 it mark it as the start.
        // When we next find it this will act as a delimiter and it can be used to
 extract that body part.
        // then the end delimiter becomes the start delimiter. and so on until the
 end delimiter is reached
        // when we break out of the loop.
        for ( int i = 0; i < body.size(); i++ )
        {
            temp = (String) body.get( i );

            if ( temp.equalsIgnoreCase( boundary ) )
            {
                if ( start == -1 )
                {
                    start = i;
                }
                else
                {
                    end = i;
                    ArrayList bodyPart = new ArrayList();
                    for ( int j = start + 1; j < end; j++ )
                    {
                        String test = (String) body.get( j );
                        bodyPart.add( test );
                    }
                    bodyPartList.add( bodyPart );
                    start = end;
                }
            }
            else if ( temp.equalsIgnoreCase( boundary2 ) )
            {
                end = i;
                ArrayList bodyPart = new ArrayList();
                for ( int j = start + 1; j < end; j++ )
                {
                    String test = (String) body.get( j );
                    bodyPart.add( test );
                }
                bodyPartList.add( bodyPart );
                start = end;
                break;
            }
        }

        /*
            Go through each of the body parts and find info on it.
        */

        // cycles through the body part list
        for ( int i = 0; i < bodyPartList.size(); i++ )
        {
            String bodyPart_content_type;
            String bodyPart_content_transfer_encoding = null;
            String bodyPart_boundary = null;

            if ( content_type.equalsIgnoreCase( "multipart/digest" ) )
            {
                bodyPart_content_type = "message/rfc822";
            }
            else
            {
                bodyPart_content_type = "text/plain";
            }

            ArrayList bodypart = (ArrayList) bodyPartList.get( i );
            ArrayList bodyPart_body = new ArrayList();
            ArrayList bodyPart_header = new ArrayList();

            for ( int j = 0; j < bodypart.size(); j++ )
            {
                String tester = (String) bodypart.get( j );
                // get header
                if ( tester.equals( "" ) )
                {
                    if ( j != 0 )
                    {
                        // copy from start to here into header
                        //(were removing from body at the same time!)
                        for ( int k = 0; k < j; k++ )
                        {
                            bodyPart_header.add( (String) bodypart.remove( 0 ) );
                        }

                        bodyPart_content_type = getMessageContentType(
bodyPart_header );
                        bodyPart_content_transfer_encoding =
getMessageContentTransferEncoding( bodyPart_header );
                        bodyPart_boundary = getMessageBoundary( bodyPart_header );
                    }
                    else
                    {
                        System.out.println( "J is equal to zero" );
                    }
                    break;
                }
            }

            parsed += parseBodyPart( bodypart, bodyPart_content_type,
bodyPart_content_transfer_encoding, bodyPart_boundary );

        }// end of outter for loop

        // collect all info back and return as one big string
        return parsed;
    }

    /**
     This method is called for all the bodyparts contained in a multipart MIME e-
mail.

     When the MIME e-mail is parsed its broken down into its constituent bodyparts.
     This method is then called for each of these bodyparts. It will call the
     relevant parsing method depending on the content-type of the body part.

     @param bodypart the bodypart that we want to parse.
     @param ct the content-type for that bodypart/
     @param cte the content-transfer-encoding for that bodypart.
     @param boundary the MIME boundary used in the bodypart.
     @return a String containing the parsed information from the bodypart.
     */
    protected String parseBodyPart( ArrayList bodypart, String ct, String cte,
String boundary )
    {
        // determine what method should be called by examining the content type.
        if ( ct.startsWith( "t" ) || ct.startsWith( "T" ) )
        {
            // if content type is text/plain then just get the body and  return it.
            if ( ct.equalsIgnoreCase( "text/plain" ) )
            {
                String parsedBody = parseTextPlain( bodypart, cte );
                return parsedBody;
            }
            // if content type is text/html parse out the html tags then return.
            else if ( ct.equalsIgnoreCase( "text/html" ) )
            {
                String parsedBody = parseTextHtml( bodypart, cte );
                return parsedBody;
            }
            // if content type is text/enriched use html parse too.
            else if ( ct.equalsIgnoreCase( "text/enriched" ) )
            {
                String parsedBody = parseTextPlain( bodypart, cte );
                return parsedBody;
            }
        }
        else if ( ct.startsWith( "mu" ) || ct.startsWith( "MU" ) )
        {
            String parsedBody = parseMultipart( bodypart, new ArrayList(), ct,
boundary );
            return parsedBody;
        }
        else if ( ct.startsWith( "me" ) || ct.startsWith( "ME" ) )
        {
            if ( ct.equalsIgnoreCase( "message/rfc822" ) )
            {
                String parsedBody = parseMessageRfc822( bodypart, cte );
                return parsedBody;
            }
        }
        return "";
    }

    /**
     This will parse e-mail message of MIME type message/rfc822.

     These e-mails are where someone has forwarded you an e-mail.
     rfc822 is the rfc first defining e-mail.

     @param body the body/bodypart containing the rfc822 e-mail.
     @param cte the content transfer encoding for the bodypart.
     @return a String containing the parsed information from this bodypart.
     */
    protected String parseMessageRfc822( ArrayList body, String cte )
    {
        ArrayList decodeBody = new ArrayList();
        ArrayList newHeader = new ArrayList();

        if ( cte != null )
        {
            try
            {
                for ( int i = 0; i < body.size(); i++ )
                {
                    String tempbuffString = (String) body.get( i );

                    byte[] encodedData = tempbuffString.getBytes();
                    ByteArrayInputStream input = new ByteArrayInputStream(
encodedData );
                    InputStreamReader bb = new InputStreamReader(
MimeUtility.decode( input, cte ) );
                    BufferedReader rr = new BufferedReader( bb );
                    String testline = rr.readLine();
                    decodeBody.add( testline );
                }
            }
            catch ( Exception e )
            {
                e.printStackTrace();
            }
        }

        // get headers ( look for first blank line)
        int firstNull = -1;
        int secondNull = -1;

        for ( int i = 0; i < decodeBody.size(); i++ )
        {
            String testcase = (String) decodeBody.get( i );
            if ( testcase == null && ( ( firstNull == -1 ) || ( secondNull == -1 ) )
)
            {
                if ( firstNull == -1 )
                {
                    firstNull = i;
                }
                else
                {
                    secondNull = i;
                }
            }
        }
        for ( int k = firstNull + 1; k < secondNull; k++ )
        {
            String testcase = (String) decodeBody.get( k );
            newHeader.add( testcase );
        }
        for ( int k = 0; k < secondNull; k++ )
        {
            decodeBody.remove( 0 );
        }

        // remove nulls from the body
        Collection c = new ArrayList();
        c.add( null );
        decodeBody.removeAll( c );

        // create a new MailMessage
        Message m = new Message();

        // get headers
        m.setHeader( newHeader );
        // get body
        m.setBody( decodeBody );
        // get Date
        m.setDate( getMessageDate( newHeader ) );
        // get From
        m.setFrom( getMessageFrom( newHeader ) );
        // get Subject
        m.setSubject( getMessageSubject( newHeader ) );
        // get content type
        m.setContentType( getMessageContentType( newHeader ) );
        // get CTE
        m.setContentTransferEncoding( getMessageContentTransferEncoding( newHeader )
);
        // get Boundary
        m.setBoundary( getMessageBoundary( newHeader ) );

        // call the main parsing again
        return parse( m );
    }


    /*
        Gets the From field out of the e-mail header
    */
    private String getMessageFrom( ArrayList headers )
    {
        for ( int i = 0; i < headers.size(); i++ )
        {
            String tester = (String) headers.get( i );
            if ( tester.startsWith( "From: " ) )
            {
                return tester;
            }
        }
        return null;
    }

    /*
        Gets the MIME Content-Type from the e-mail header
    */
    private String getMessageContentType( ArrayList headers )
    {
        for ( int i = 0; i < headers.size(); i++ )
        {
            String tester = (String) headers.get( i );
            if ( tester.startsWith( "Content-Type: " ) )
            {
                int start = tester.indexOf( ":" );
                int finish = tester.indexOf( ";" );
                if ( finish == -1 )
                {
                    String tester2 = tester.substring( ( start + 2 ) );
                    return tester2;
                }
                else
                {
                    String tester2 = tester.substring( ( start + 2 ), finish );
                    return tester2;
                }
            }
        }
        // if none present its defaulted to text/plain
        return "text/plain";
    }

    /**
     Gets the Content-Transfer-Encoding field from the e-mail header.
     */
    private String getMessageContentTransferEncoding( ArrayList headers )
    {
        for ( int i = 0; i < headers.size(); i++ )
        {
            String tester = (String) headers.get( i );
            if ( tester.startsWith( "Content-Transfer-Encoding:" ) )
            {
                int start = tester.indexOf( ":" );
                tester = tester.substring( ( start + 2 ) );
                return tester;
            }
        }
        return null;
    }

    /**
     This method will take the Subject line from the header and remove
     the "Subject: " from the start of it.

     @param headers the arraylist of header fields.
     @return String the subject or null if there is no subject.
     */
    private String getMessageSubject( ArrayList headers )
    {
        for ( int i = 0; i < headers.size(); i++ )
        {
            String tester = (String) headers.get( i );
            if ( tester.startsWith( "Subject: " ) )
            {
                String theSubject = tester.substring( 9 );
                return theSubject;
            }
        }
        return null;
    }

    /**
     This method will take the Date header field from the headers and remove the
     "Date: " from the start of it.

     @param headers the ArrayList of header fields.
     @return String the Date (in text format) or null if there was no date.
     */
    private String getMessageDate( ArrayList headers )
    {
        for ( int i = 0; i < headers.size(); i++ )
        {
            String tester = (String) headers.get( i );
            if ( tester.startsWith( "Date: " ) )
            {
                return tester;
            }
        }
        return null;
    }


    /*
        Gets the MIME Boundary from the e-mail header.
    */
    private String getMessageBoundary( ArrayList headers )
    {
        for ( int i = 0; i < headers.size(); i++ )
        {
            String tester = (String) headers.get( i );
            int a = tester.indexOf( "boundary=\"" );

            if ( a != -1 )
            {
                int b = tester.indexOf( "\"", ( a + 12 ) );

                String boundary = "--" + tester.substring( ( a + 10 ), b );
                return boundary;
            }
        }
        return null;
    }

    /*
```

```
        This method will extract just the hostname from a URL
    */
    private String getUrl( String s )
    {
        // find index of "//"
        // find index of "/" or end of word
        int i = -1;
        int j = -1;

        i = s.indexOf( "//" );
        j = s.indexOf( "/", ( i + 2 ) );

        if ( i != -1 )
        {
            if ( j != -1 )
            {
                String temp = s.substring( ( i + 2 ), j );
                temp = "URL:" + temp;
                return temp;
            }
            else
            {
                String temp = s.substring( ( i + 2 ) );
                temp = "URL:" + temp;
                return temp;
            }
        }
        return "URL:" + s;
    }
}
```