Chapter IV

Facilitating PubMed Searches: JavaServer Pages and Java Servlets

Introduction

J2EE is a powerful platform for developing sophisticated web-based applications. This *J2EE* feature is especially critical for Bioinformatics software development given the availability of a large number of important biological sequence and biomedical data repositories on the WWW that biologists need to access on a routine basis for their research. We will explore one such resource - NCBI PubMed - in detail in this Chapter and introduce the *Java Servlet* and *JavaServer Pages (JSPs)* technologies to facilitate searching, retrieval and storage of biomedical data from PubMed.

HTTP and CGI

We will begin by refreshing our basic knowledge of standard protocols such as the Hypertext Transfer Protocol (HTTP) and the Common Gateway Interface (CGI) that allows for a server to pass requests from a client web browser to an external application and in return allow the web server to return the output from the application to the web browser. Although there are several more HTTP commands than GET and POST, we will introduce only these methods here and refer interested readers to the HTTP specification Request for Comments 2616 (RFC 2616) for more information.

HTTP Protocol

HTTP is a client/server protocol that WWW users utilize everyday to download web pages to their web browsers. The client part of this protocol is handled by the web browser that sends a request to the server (also called an HTTP server or a web server). The server responds to the request with a web page. That, put very simply is all that HTTP does, at least for the purpose of this discussion.

The request sent by the client contains an HTTP command with a set of parameters that define the request. For example, to request an HTML document called index.shtml from the NCBI server, one can issue the following command using telnet:

```
telnet www.ncbi.nlm.nih.gov 80
GET /blast/index.shtml HTTP/1.0
```

telnet is a program that connects a local computer to a server on the network and allows users to issue commands directly to the remote server. The HTTP protocol works over the Transmission Control Protocol/Internet Protocol, a suite of communications protocols used to connect hosts on the WWW, also called TCP/IP for short. In this case, the HTTP protocol works over the TCP/IP protocol that one can access through a session initiated by telnet, using the specified server address (www.ncbi.nlm.nih.gov) and the port (80).

There are other pieces of information that could be passed to the request, to specify information about the client and the type of data it would like to receive. Also a blank line specifying the end of the request must be added at the end.

When such a request is sent to the NCBI server, the output received contains several difference bits of data, along with the actual document requested, if found.

```
HTTP/1.1 200 OK
Date: Sun, 12 Feb 2006 18:13:42 GMT
Server: Nde
Accept-Ranges: bytes
Content-Type: text/html
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE    html    PUBLIC    "-//W3C//DTD    XHTML    1.0
```

```
Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
```

(The output has been truncated for clarity.)

The first line corresponds to a code indicating the status of the response - 200 OK - which means the requested operation was executed successfully. After the status line we have information about the server itself. Finally if the document is available it is sent within the rest of the response.Other code and associated descriptions are defined in the HTTP specification and provide information regarding any problems accessing the server, if the requested document is not found, etc.

GET and POST Methods

Although a client can send different HTTP commands, the GET and POST commands are the most commonly used. GET allows users to retrieve or get information from an HTTP server, while the POST HTTP command allows users to post or send information to the server. The POST information resides on the server, usually within a database. The GET command is just for querying the HTTP server and therefore won't be stored, unless for statistical purposes or for logging the load on the server.

GET can send parameters within the body of the URL to specifically query the HTTP server. Since GET was designed for querying purposes, the URL length is limited to a certain number of characters (250) on certain servers. The POST method, on the other hand, can send more information, including different documents types, and does not have a constraint on length.

CGI For Generating Dynamic Content

According to RFC 3875, CGI is a

```
"... simple interface for running external programs,
software or gateways under an information server in a
platform-independent manner."
```

This simply means that if you have a program that runs on your Unix machine and you want to access it through a web browser, you can do so using CGI. The way it works is that each time you request to run that program, the web server will create an instance of the program, pass to it all the parameters obtained from the request that was sent, wait for the program to process the information and then wrap the program output into an HTTP response.

This allows users to generate the content of a web page dynamically instead of accessing static HTML content. It can be very slow when 100 users access the same program because the server must create 100 instances of the same program to run the 100 queries.

A number of vendors have implemented their own API's to handle the performance issues of CGI or to replace that interface with proprietary protocols. Sun Microsystems, for example, has developed proprietary technology that will run in a Java Virtual Machine and handle the required processes that live on the server via the *Servlets* and *JavaServer Pages* technologies.

Servlets and JavaServer Pages Technologies

Now that we're more familiar with HTTP, it's time to learn about *servlets* and *JSPs*. Before we present the Java API, lets briefly review the advantages of using *servlets* over typical CGI programs:

- Once the *servlet container* is started, each *servlet* runs in the same process as the container; this avoids creating new processes for each request, unlike CGI programs.
- Because the *servlet* is created once at startup, it remains in memory and there is no overhead associated with loading the Java class multiple times. The service just needs to request the *servlet* from a pool and call its service method.
- A *servlet* is reusable, which saves memory and time.

These characteristics allow faster execution of the server processes to generate dynamic content. In addition, the fact that it is Java brings with it the power of the "Write once, run everywhere" properties of the platform.

Java API for Servlets and JSPs

From the *servlet* specification available at the Sun Microsystems website, a *servlet* is defined as a "Java technology-based Web component, managed by a container, that generates dynamic content". *Servlets* are Java classes that implement a base interface called `Servlet`, from the `javax.servlet` package available in the *Java Enterprise Edition Platform.* `javax.servlet.Servlet` is the basic interface which provides the `service()` method that handles a client request independently of the protocol used to communicate between the client and the server. To create a *servlet* one can directly implement this interface or extend `GenericServlet` or `HttpServlet`.

The life cycle of a servlet is managed through three methods:

- `init`: the container instantiates a `Servlet` object and calls init to initialize it.
- `service`: upon a client request, the container get the *servlet* and calls its `service` method.
- `destroy`: when the *servlet* is not in use any more, the container will call the `destroy` method.

Fig. 4.1 below shows the life cycle of a servlet (called MyServlet) when a client request comes to the container.

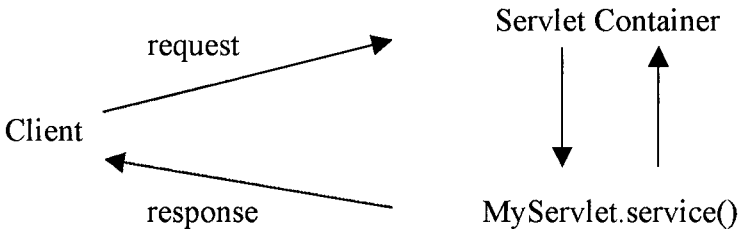

Fig. 4.1. Life cycle of servlets

Since we want to deal with HTTP requests, we are primarily interested in the `javax.servlet.http.HttpServlet` package to create `HttpServlet` Java classes. We will learn more about this package in the next few sections.

***

Before we delve into the servlet and JSP technologies, let's briefly review the MVC framework that we had introduced in Chapter 1, which we will be using as a guiding principle for building our web application. We will also briefly review the *Apache Tomcat Server*, which we will use as our *servlet container*. Finally we will also talk briefly about the *JavaServer Pages Standard Tag Library (JSTL),* to introduce the concept for the benefit of readers to explore further on their own.

JavaServer Pages Standard Tag Library (JSTL)

JavaServer Pages (JSPs) use *custom tags* to perform all kinds of manipulations like iterating over collections, transforming one object into another, form processing, database access, and the like. The idea behind *JSTL* is to create libraries with *reusable tags*. These tags can be used and customized like functions or methods in Java. This also creates clarity in the *JSP* file because the *tags* allow users to keep the *JSP* as the *View* and the *business logic* or the *Controller* and the *Model* separated from each other. In other words, one can think of *JSTL* as a Java package that groups together functionalities into a set of independent and reusable and tags.

Apache Tomcat Server

Tomcat is an open source *servlet container*, which implements the Java *Servlet* and *JavaServer Pages* technologies written in Java. This is the *servlet container* we will be using in this Chapter. The Tomcat servlet container allows developers to deploy web applications as well as to monitor and manage them. Tomcat compiles the *JSPs* into *servlets* when first called, or just before calling the application. Tomcat also allows defining the *realm* for specific authentication and authorization services that may be required for web applications. A "realm" in Apache terminology is "a "database" of usernames and passwords that identify valid users of a web application (or set of web applications), plus an enumeration of the list of roles associated with each valid user." The reader is referred to the Appendix for further information on how to install Tomcat. More information can also be found at the Apache Tomcat Project website of The Apache Software Foundation.

The NCBI PubMed Literature Search and Retrieval Service

PubMed is a resource maintained by the National Library of Medicine (NLM), under the aegis of the National Center for Biotechnology Information (NCBI, National Institutes of Health, USA) and provides access to over 14 million citations for biomedical articles dating back to the 1950's. PubMed is a vast resource and covers scientific findings from a diverse array of disciplines including but not limited to the natural and physical sciences. According to usage statistics from NCBI, over 59,000,000 queries seeking scientific information were submitted to the PubMed server in March 2004 alone (http://www.ncbi.nlm.nih.gov/About/tools/restable_stat_pubmed.html). Indeed, PubMed is an indispensable resource for researchers all over the world.

As vast and valuable as PubMed is, average users still have to contend with the problem of retrieving useful and relevant knowledge from the underlying database in a piecemeal fashion using one or more keywords. PubMed also doesn't currently provide a way to intelligently or visually analyze the results of a query (for example, by highlighting or color coding the search terms in a retrieved abstract, etc). We will address some of these issues and create solutions for them in this Chapter to enhance the value of literature search and retrieval through PubMed.

Accessing Biomedical Literature Through Entrez

Access to information in NCBI databases is granted through a service called Entrez, a search and retrieval system maintained by NCBI that combines information on individual DNA and protein sequences, large-scale sequence data from whole genomes, and information on 3-dimensional structures of biomolecules. It also grants access to MEDLINE, which covers research in a number of Life Science areas such as medicine, nursing, dentistry, veterinary medicine, the health care system, and preclinical sciences. The steps involved in a typical search on PubMed are described below. We will use the generic keyword "HIV" (for Human Immunodeficiency Virus, the causative agent of Acquired Immune Deficiency Syndrome, AIDS) for the illustration.

Step 1: User navigates to the NCBI PubMed website (**Fig. 4.2**):

http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed

Step 2: User enters the search term 'HIV' (the search is case-insensitive) in the search box and presses Enter. PubMed presents the user with a list of citations relevant to the search term (**Fig. 4.3**). Internally, PubMed searches for a match between the supplied keyword(s) and terms in the Medical Subject Headings (MeSH) Translation Table, an alphabetical hierarchy of controlled vocabulary terms used for subject analysis of biomedical literature at the NLM. The list of citations may span several thousand pages depending on the number of articles that match the search term. Each journal article on PubMed is associated with a unique numeric tag called the PubMed Unique Identifier or PMID.

Step 3: User clicks on the citation to display specific information (Brief, Abstract, Medline etc) about each journal article (**Fig. 4.4**) or selects several articles to display (**Fig. 4.5**).

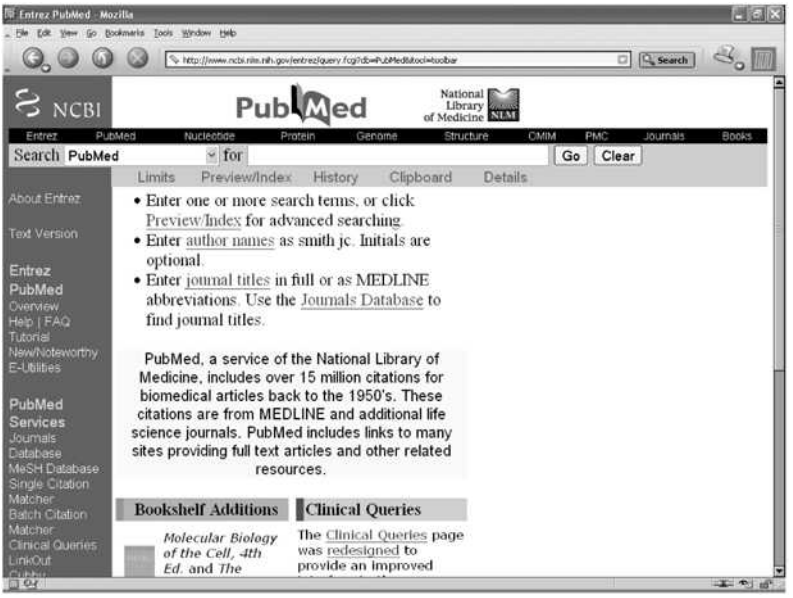

Fig. 4.2. The NCBI PubMed web resource

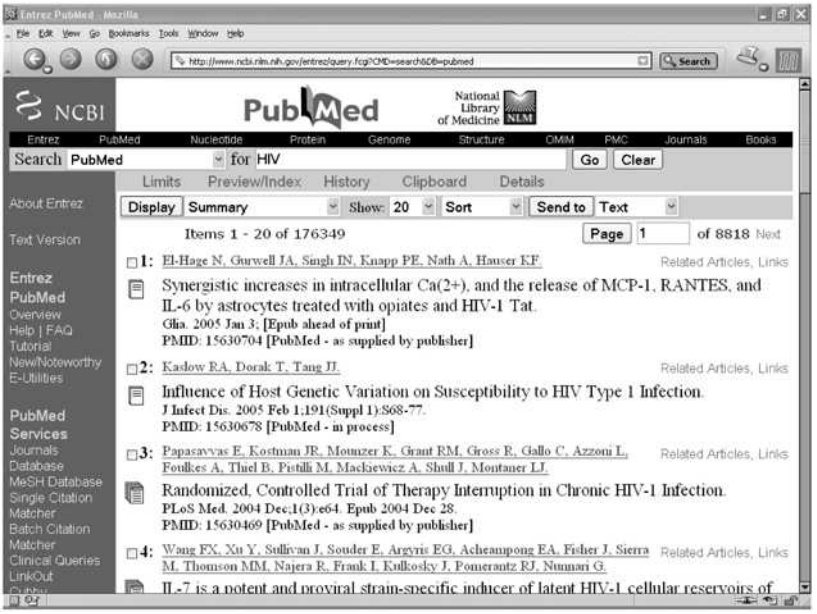

Fig. 4.3. Search results for the term 'HIV'

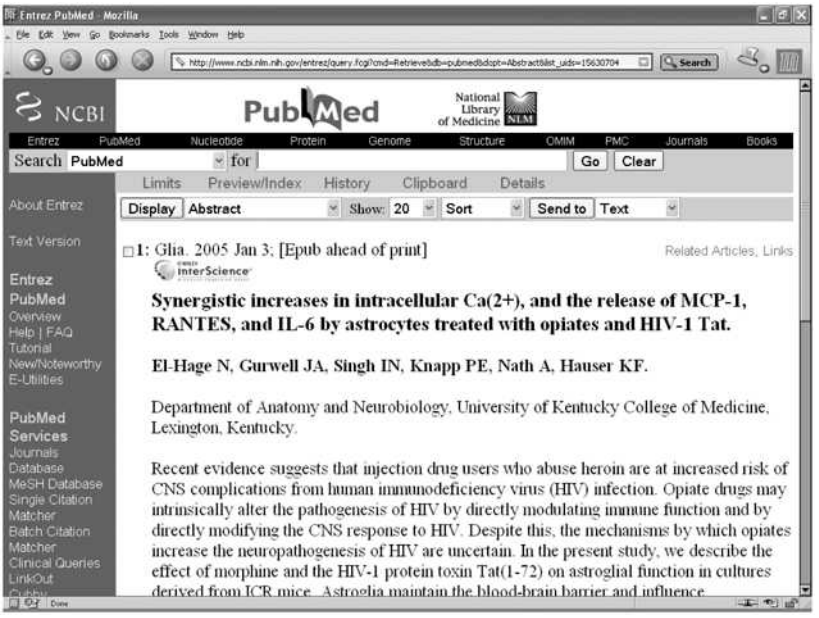

Fig. 4.4. Viewing abstracts for individual journal articles

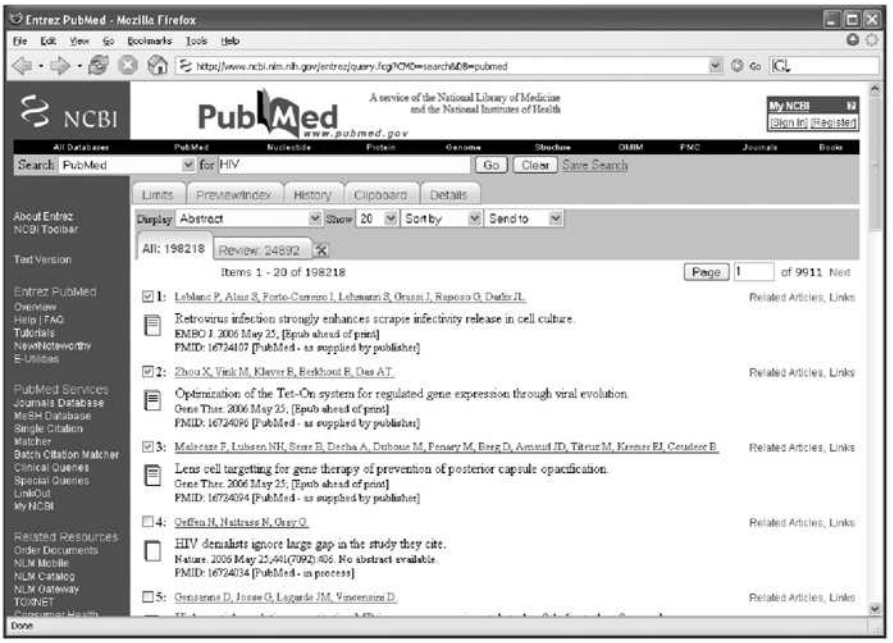

Fig. 4.5. Selecting several articles to view abstracts

The user can save articles of choice in the chosen display format (Summary, Abstract, etc) by selecting the required articles and pressing the "Send to" button and selecting the appropriate format (Text, File, Email, etc) (**Fig. 4.6**).

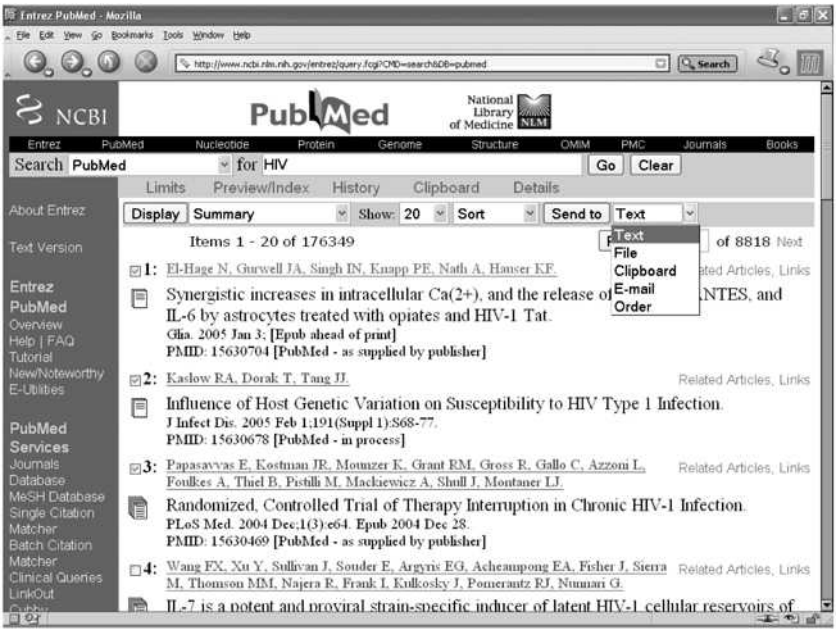

Fig. 4.6. Saving search results for selected abstracts

The search process quickly becomes unwieldy especially when information from a large number of citations needs to be extracted and analyzed. In this Chapter, we will demonstrate the power of Sun's JavaServer Pages and Java Servlets technologies to build a web-based application to simplify the process of accessing information on PubMed. We will use the Apache Tomcat server as the servlet container and the Apache Ant tool to build and deploy the Java web-based application. Please refer to the Appendix to download the tools and for instructions on using them.

Create Web Application With Servlets and JSPs

Servlets as we described earlier are Java code that run on a server and provide a general framework for services built using the request-response paradigm. HTTP, is one such paradigm that is implemented through the *javax.servlet.http* package from the Java Servlet API. On the other hand, *JSPs* were designed to mainly allow the separation of the business logic (what the application does) from the appearance of the page (how the application displays the result).

The steps and the flow diagram below illustrate the behavior of such an application (**Fig. 4.7**):

Step 1: The user accesses the application through a web browser. The actual code that runs the application remains hidden from view. The user only sees and interacts with an HTML page, which for our first application will contain a simple search form consisting of a single text-box and a submit button. The user enters a single keyword (search term) in the text-box and presses the submit button. After the search is processed by the application, the user sees the results in the web browser. **Fig. 4.7** illustrates the actions of the user in the User Space.

Step 2: The application is implemented as a *servlet* that gets the information entered on the search form and processes the request on the NCBI PubMed server. This involves a series of operations. The application constructs the PubMed URL that is specific to the entered search term. Next, through a URL object, it sends a request to the PubMed server. The PubMed server performs the search using the keyword and formulates a response, which is an HTML document containing a list of citations matching the search term. These operations are shown in the Application Space (**Fig. 4.7**).

Step 3: After processing the request, the PubMed server sends the search results back; the application reads the result from the URL using a `BufferedReader` object to retrieve the content sent back from the server.

Step 4: Once the response is received, the application reads the contents of the response using the `BufferedReader` object and prints it out to the screen using the `javax.servlet.http.HttpServletResponse` object.

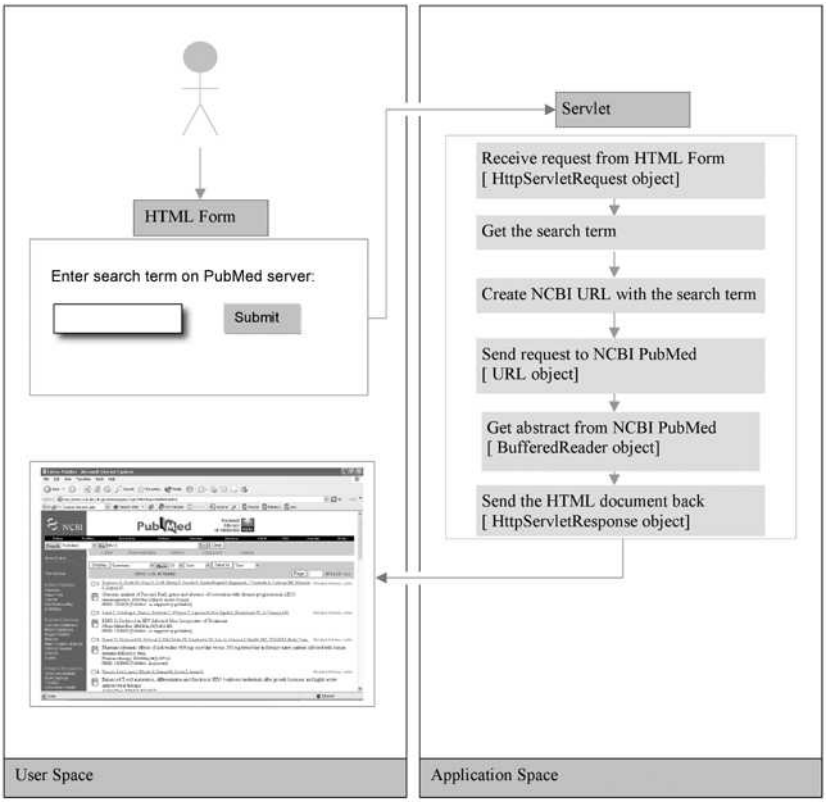

Fig. 4.7. The structure of the PubMed1 servlet

Web Application Structure

According to the Java Servlet API Specification 2.2 (http://java.sun.com/products/servlet/download.html), a web application (or web app) is a collection of *servlets*, HTML pages, classes, images, and other resources that can be bundled and run on multiple containers from multiple vendors. Simply stated, a web app bundles resources together to provide a portable and server independent way to access information via a web browser. In order to be portable and server independent, a web app must be designed according to a well-defined schema that dictates where the resources used by the web app are to be placed. This ensures that there is no conflict between the different resources used by the web app. The web app has to be installed on the web application server and mapped to a specific uniform resource identifier (URI) path (called also the *servlet*

context path) on the server. The file structure of the web app is archived into a WAR file (Web application ARchive).

For example, the application we are writing is installed on the web app server using the path pubmed, for example:

```
http://localhost:8080/pubmed
```

as is explained further below. Here is the file structure of the `pubmed` web app being developed:

```
example.html
pubmedExample.jsp
jsp/moreSpecificPubmedExample.jsp
pics/pubmedLogo.png
anotherLogoExample.png
WEB-INF/web.xml
WEB-INF/classes/servlet/DataRetriever.class
WEB-INF/lib/Jakarta-regexp-1.3.jar
```

The basic layout that defines a web app file structure is as follows:

- HTML, JSP, PNG (image) and other resource files must be located in the root directory to be visible in the web browser.
- *web.xml* is located in the *WEB-INF* directory under root. *web.xml* is the Web Application Deployment Descriptor for the application. This file defines in an XML format the configuration information utilized by the web app such as initialization parameters, *servlet* mappings, security constraints, etc.
- *WEB-INF/classes*: This directory contains all the Java classes (and *servlets*) with any resources associated with them that make the web app. The Java class `servlet.DataRetriever` is stored in `WEB-INF/classes/servlet/DataRetriever.class`.
- *WEB-INF/lib*: This directory contains all the Java™ Archive (JAR) files required to run the web app, including third parties libraries such as Jakarta-regexp-1.3.jar for regular expression matching.

Access to the web app or any resource from the web application server available at the localhost and port 8080 is through the following URL:

```
http://localhost:8080/pubmed
```

This access is set up in the *http.conf* configuration file located in the Tomcat '*conf*' directory. Any web application is deployed on the web application server using a relative path.

If we want to access the HTML pages located in the WAR file in the root directory, for instance, for a file called `example.html`, we open the following URL in the web browser:

```
http://localhost:8080/pubmed/example.html
```

The WAR archive may also contain images that can be found in the `/pics` directory. To access the `pubmedLogo.png` picture, for example, we need to point our web browser to the following URL:

```
http://localhost:8080/pubmed/pics/pubmedLogo.png
```

To access the *servlet* `DataRetriever`, in the web app descriptor file we wrote the mapping from the path in the URL to the actual Java class that is going to handle the HTTP request. This Servlet can be accessed at the following path:

```
http://localhost:8080/pubmed/DataRetriever
```

Creating a Servlet to Access Biomedical Literature

We begin by declaring a package called `PubMed`. Next, we import the necessary packages, which contain the classes that are used by the *servlet*. In order to implement the design described in **Fig. 4.7**, we need to create a Java `Servlet` class called `PubMedServlet1_1` that extends `javax.servlet.http.HttpServlet`, the standard base class for HTTP *servlets*. We then need to override the `doGet()` method as shown in the code below. The `doGet()` method takes two parameters: the `HttpServletRequest` object (called req) which is the client request and an `HttpServletResponse` object (called res) which is the response sent back to the client. Since the method returns nothing, its return type is *void*.

It is conceivable that the process of sending a request to a remote server and obtaining a response back may encounter errors. Java has objects called *Exceptions* to handle such occurrences. The Java Virtual Machine (JVM) will inform the caller using *Exception* objects when a program does not behave the way it is supposed to do. This object is "*thrown*" when that error or unusual condition occurs and it stores information about the

particular error event. In order to inform the developer that such an *exception* can be "thrown" from the method, we use the appropriately named "*throws*" Java keyword in the method signature. We declare `ServletException`, which defines a general *exception* a *servlet* can throw when it encounters errors and `IOException` to catch errors due to failed or interrupted I/O operations. Another way to handle *exceptions* is to use the *try-catch block*. We will see how to use *try-catch blocks* later in the Chapter.

Let's return to the servlet creation process. The general signature of the `doGet` method is shown below:

```
    Protected void
doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException { }
```

since we are sending a text or HTML response, we set the content type to text/html with the line:

```
res.setContentType("text/html");
```

Next we request a `PrintWriter` object to write the text to the response message:

```
PrintWriter out = res.getWriter();
```

Next we create HTML to create a form that users can utilize for conducting searches on PubMed. In its simplest state, the form will have a title, a search box and a submit button. The HTML for the form is as follows:

```
<HTML>
  <HEAD><TITLE>PubMed Servlet 1.1</TITLE></HEAD>
  <BODY>
    <b>Java for Bioinformatics: </b>
      <font     color=red><b>PubMed     Servlet     version
1.1</b></font>\n
      <BR><BR><B>Please enter a term to search on NCBI
PubMed:</B><BR><BR>\n
      <FORM METHOD=GET>\n
      <INPUT TYPE=TEXT NAME=searchTerm><BR><BR>\n
      <INPUT TYPE=SUBMIT VALUE=\"Search PubMed\"><BR>\n
    </FORM>
  </BODY>
</HTML>
```

The search form as it appears in a browser is shown in **Fig. 4.8**.

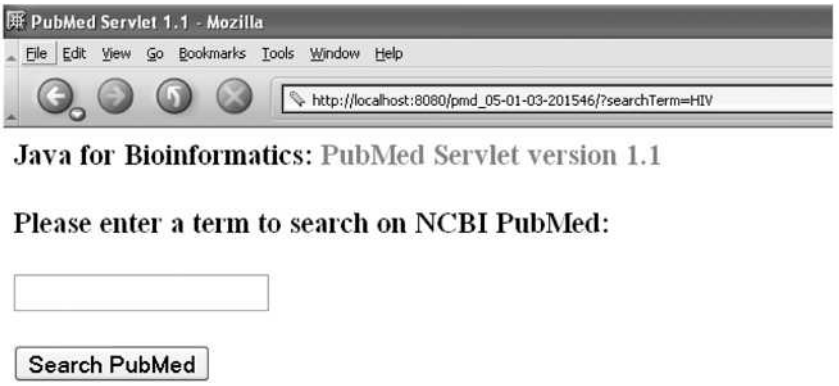

Fig. 4.8. The PubMed servlet version 1.1 search form

To implement the form in code, we create an object called html of the type `StringBuffer`:

```
StringBuffer html = new StringBuffer("<HTML>");
```

and append the HTML code to it:

```
StringBuffer html = new StringBuffer("<HTML>");
      html.append("<HEAD><TITLE>PubMed Servlet
1.1</TITLE></HEAD><BODY>\n");
      html.append("<b>Java for Bioinformatics: </b>");
      html.append("<font color=red><b>PubMed Servlet version
1.1</b></font>\n");
      html.append("<BR><BR><B>Please enter a term to search on NCBI
PubMed: </B><BR><BR>\n");
      html.append("<FORM METHOD=GET>\n");
      html.append("<INPUT TYPE=TEXT NAME=searchTerm><BR><BR>\n");
      html.append("<INPUT TYPE=SUBMIT VALUE=\"Search
PubMed\"><BR>\n");
      html.append("</FORM>\n");
```

The URL to send the search term is:

```
http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?dispmax=10&db=pubmed&
cmd=search&term=term
```

In code we implement this in the following manner:

```
  URL url = new URL
("http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?dispmax=10&db
=pubmed&cmd=search&term=" + URLEncoder.encode(term, "UTF-
8"));
```

Note the parameters on the URL (separated by ampersand symbols '&') that specifies what information we want to submit to the PubMed engine to retrieve data:

```
dispmax=10
db=pubmed
cmd=search
term=term
```

We are limiting the search to ten articles (dispmax=10) for the purpose of illustration only. We select the database as PubMed (db=pubmed) and provide the command to search (cmd=search) with the search term (term=term). Next, we open the connection to the server:

```
  URLConnection urlConnection = url.openConnection();
  BufferedReader reader = new BufferedReader(new
InputStreamReader
      (urlConnection.getInputStream()));
```

In the next step, we construct a regular expression to extract the PubMed Ids (PMIDs) of the abstracts that match the search term and create an array to store them. To do this, we will use a Java Regular Expression package available from The Apache Jakarta Project available as a JAR file called jakarta-regexp-1.3.jar:

```
String s = null;
RE pmidRE = new RE("PMID: ([0-9]+) \\[PubMed");
Collection pmids = new ArrayList();

while ((s = reader.readLine()) != null) {
  if (pmidRE.match(s)) {
    pmids.add(pmidRE.getParen(1));
  }
}
reader.close();
```

Listing 4.1 shows the code for PubMed servlet version 1.1

Listing 4.1. PubMed Servlet version 1.1

```
package org.jfb.PubMed;
import org.apache.regexp.RE;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;
import java.net.URL;
import java.net.URLEncoder;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.Properties;

public class PubMedServlet1_1 extends HttpServlet {
  protected void doGet(HttpServletRequest req,
HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    StringBuffer html = new StringBuffer("<HTML>");
    html.append("<HEAD><TITLE>PubMed Servlet
1.1</TITLE></HEAD> <BODY>\n");
    html.append("<b>Java for Bioinformatics: </b>");
    html.append("<font color=red><b>PubMed Servlet version
1.1</b></font>\n");
    html.append("<BR><BR><B>Please enter a term to search
on NCBI PubMed:</B> <BR><BR>\n");
    html.append("<FORM METHOD=GET>\n");
    html.append("<INPUT TYPE=TEXT
NAME=searchTerm><BR><BR>\n");
    html.append("<INPUT TYPE=SUBMIT VALUE=\"Search
PubMed\"><BR>\n");
    html.append("</FORM>\n");

    String term = req.getParameter("searchTerm");
    if (term != null) {
      html.append("<BR><HR><BR>");
      html.append("You have searched NCBI for the term
<font color=red>'" + term + "'</font>.");
      URL url = new
URL("http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?dispmax=10
&db=pubmed&cmd=search&term=" + URLEncoder.encode(term, "UTF-
8"));
      URLConnection urlConnection = url.openConnection();
      BufferedReader reader = new BufferedReader(new
InputStreamReader(urlConnection.getInputStream()));

      String s = null;
```

```
        RE pmidRE = new RE("PMID: ([0-9]+) \\[PubMed");
        Collection pmids = new ArrayList();

        while ((s = reader.readLine()) != null) {
          if (pmidRE.match(s)) {
            pmids.add(pmidRE.getParen(1));
          }
        }
        reader.close();

        html.append("<BR><br>PMIDs found:<br>\n");
        int i = 1;

        for (Iterator iterator = pmids.iterator();
iterator.hasNext();) {
          String s1 = (String) iterator.next();
          html.append("<a href=\"")

.append("http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Re
trieve&db=pubmed&dopt=Abstract&list_uids=")
            .append(s1)
            .append("\">")
            .append(s1)
            .append("</a>\n");
          if (iterator.hasNext() && i++ != 5) {
            html.append(" - ");
          } else {
            html.append("<BR>");
          }
        }
      }
      html.append("</BODY></HTML>\n");
      out.print(html.toString());
    }
  }
```

The next few lines of code iterate over the array for each of the PMIDs of abstracts matching the search term and print them out along with a hyperlink to the original abstract on PubMed. The structure of the servlet and its component files is shown below (**Fig. 4.9**).

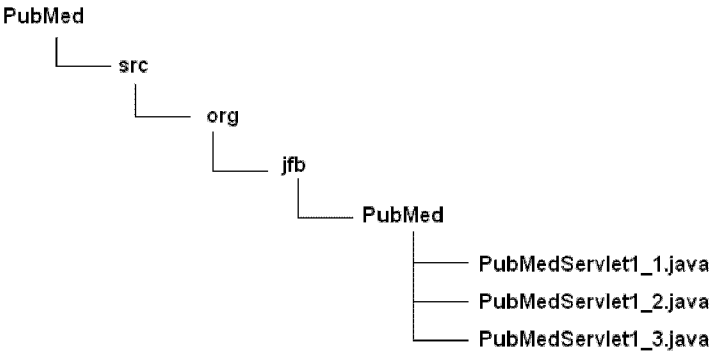

Fig. 4.9. The PubMed servlet structure

To see the *servlet* in action, start the Apache Tomcat Server, compile the code and run it with the command:

```
ant install
```

Apache Ant is a Java-based build tool used to manage the different steps in the development cycle of an application, which include compilation of the code libraries needed for the application, creating the necessary JARs for deploying an application, etc. It is available from The Apache Software Foundation website. For further information on installation and use, please refer to the Appendix.

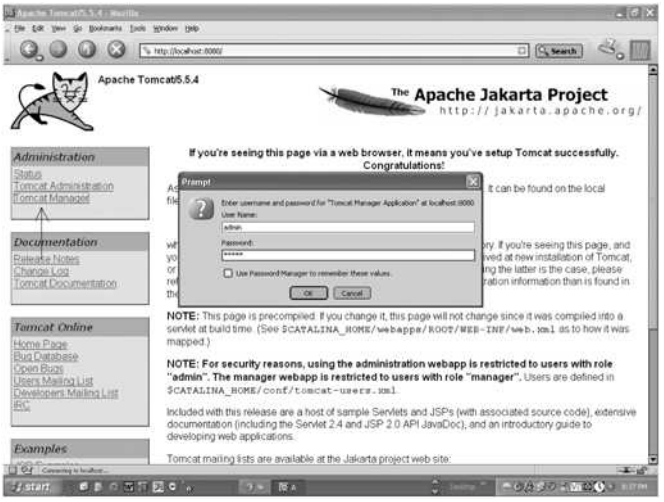

Fig. 4.10. Logging into the Tomcat Manager

Open the following URL:

```
http://localhost:8080
```

When the Apache Tomcat welcome page loads, click on the Tomcat Manager visible on the left panel and login into the server using the credentials you specified during installation (**Fig. 4.10**). Access the latest build of the application to view the servlet. The output of the search with the keyword HIV using the first version of our program, which we will call PubMed Servlet version 1.1, is shown in **Fig. 4.11**.

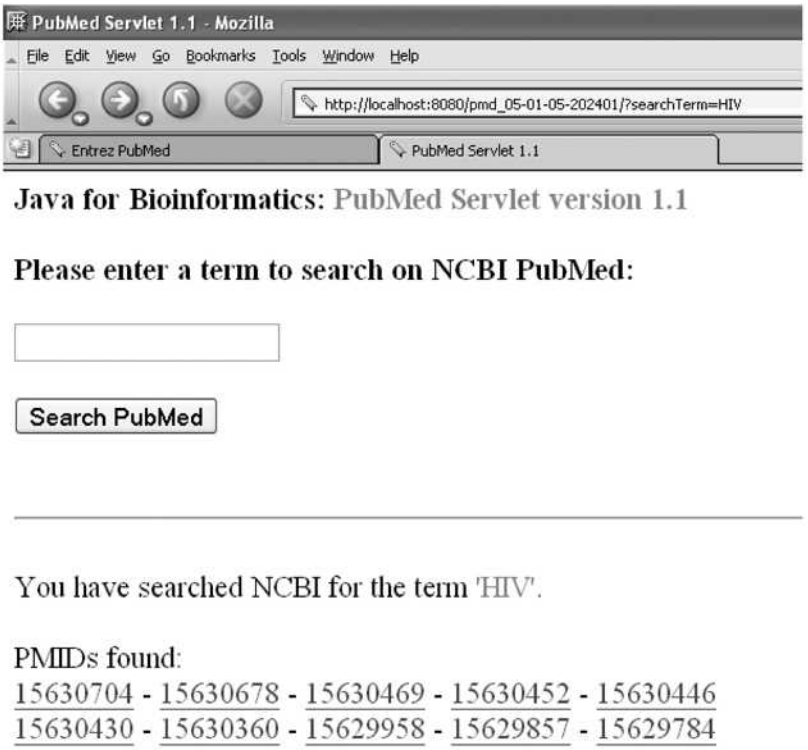

Fig. 4.11. Output from the PubMed servlet using search term "HIV"

In the first version of the application, we are simply validating our approach and displaying just the PMIDs for the abstracts that match the entered keyword. To check that the code works and retrieves the correct data, we hyperlink the PMIDs to the original abstracts on PubMed. Clicking on 15630704, for example, opens up the abstract corresponding

to the PMID for the abstract that shows up in the search performed directly on the NCBI PubMed webpage (**Fig. 4.12**).

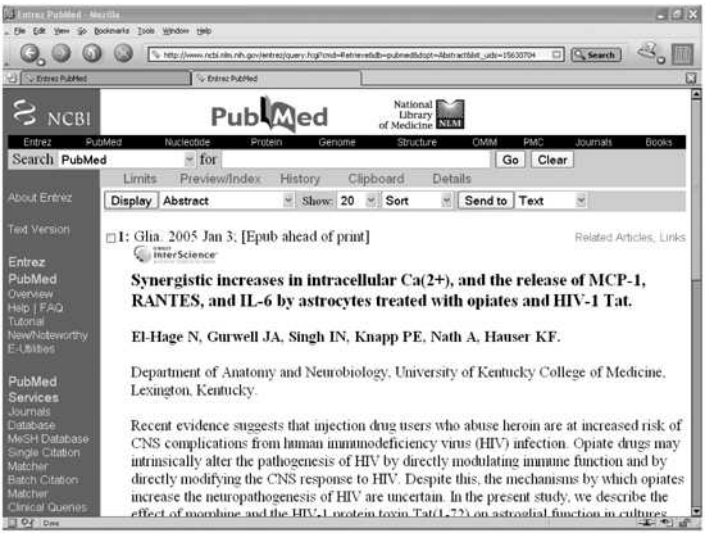

Fig. 4.12. PubMed article corresponding to PMID 15630704

The results in **Fig. 4.11** and **Fig. 4.12** are identical to the search output obtained from a search with the keyword 'HIV' at NCBI PubMed at the time of this writing (**Fig. 4.13**).

Fig. 4.13. Results of NCBI PubMed search with keyword "HIV"

Displaying PubMed Abstracts

In order to make the search output more useful for researchers, we would like to parse the abstract from each citation and make it available for viewing right up front as part of the search results. We will now create the code to parse out the abstract from each of the articles that are returned by a search.

The general framework of the program is as follows:

1. Create the search form
2. Retrieve the keyword(s) provided by the user
3. Retrieve PMIDs from PubMed corresponding to the search term
4. Retrieve abstracts based on each of the PMIDs obtained in step 1
5. Iterate #4 until all abstracts have been retrieved

To create the search form, we create a method called `createSearchForm()` which creates a variable of type *StringBuffer* called html and append the various html tags to it in succession:

```
    private StringBuffer createSearchForm() {
      StringBuffer html=new StringBuffer();
      html.append("<HTML>");
      html.append("<HEAD><TITLE>" + TITLE +
"</TITLE></HEAD><BODY>");
      html.append("<b>Java for Bioinformatics: </b>");
      //html.append("<font color=red><b>PubMed Servlet
version 1.1</b></font>\n");

      html.append("<font color=red><h1>" + TITLE +
"</h1></font>");
      //html.append("<B>Please enter a userKeywords to search
on NCBI:</B><BR><BR>\n");
      html.append("<BR><B>Please enter a term to search on
NCBI PubMed: </B><BR><BR>\n");

      html.append("<FORM METHOD=GET>\n");
      html.append("<INPUT TYPE=TEXT NAME=" + KEYWORDS +
"><BR><BR>\n");
      html.append("<INPUT TYPE=SUBMIT VALUE=\"Search
PubMed\"><BR>\n");
      html.append("</FORM>\n");
      return html;
    }
```

Note that the text box for entering keywords is called KEYWORDS. We will use this name to retrieve the user-supplied keywords. The resulting

search form for the next iteration of the application, which we will call PubMed Servlet version 1.2, is shown in **Fig. 4.13**.

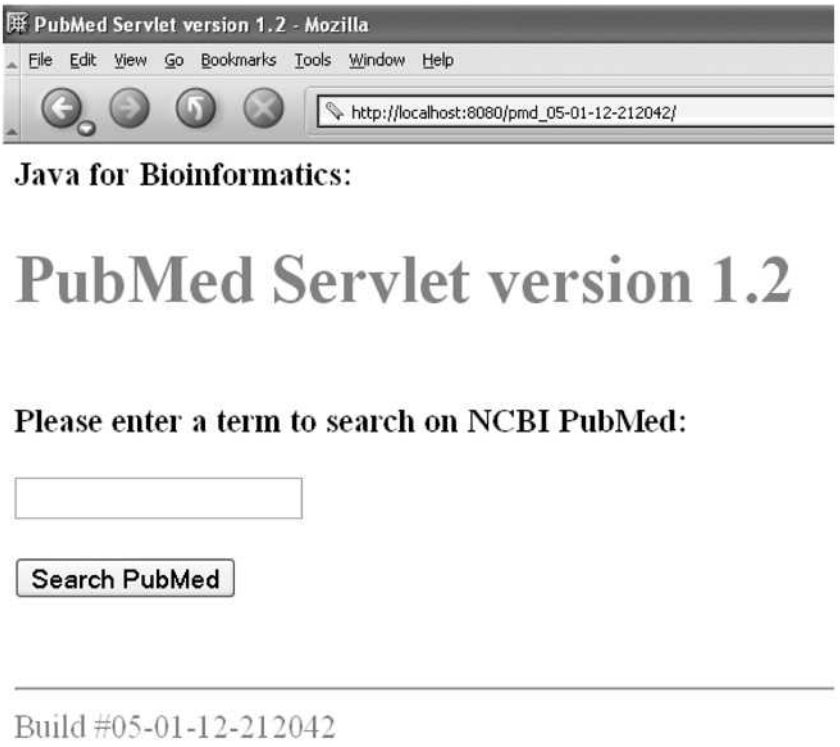

Fig. 4.13. PubMed servlet search form version 1.2

We then retrieve the keyword(s) from the search box using a method called `getUserKeywords()`:

```
String userKeywords = getUserKeywords(req);
```

This method takes in the *HttpServletRequest* req object as a parameter to return the keywords:

```
private String getUserKeywords(HttpServletRequest req) {
    return req.getParameter(KEYWORDS);
  }
```

The next few lines perform some basic user input validation. If you press the search button without supplying any keywords, for example, the

program will return an error message: "Please enter keywords to search." (**Fig. 4.14**).

Fig. 4.14. User-input validation

We then create a variable of type *StringBuffer* called `sbPmids` to store PMIDs corresponding to the search terms and a *String* variable called `searchURL` to specify the search URL:

```
  StringBuffer sbPmids = null;
       final String searchURL =
"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?dispmax=10&db=
pubmed&cmd=search&term=" + URLEncoder.encode(userKeywords,
"UTF-8");
```

Next we write a method called `getPmids()` to retrieve PMIDs from the keywords. The method takes one parameter, the `searchURL`, which in turns

contains the keyword(s) embedded in it. The result of the operation is stored in an object called `sbPmids`:

```
sbPmids = getPmids(searchURL);
```

We place the method within a *try-catch block* we had briefly mentioned earlier to catch any *exceptions* that may arise while the request is sent to PubMed. If we do indeed encounter an exception, the method will trap the error, print out the offending error message and exit.

```
  try {
    sbPmids = getPmids(searchURL);
  }
  catch (IOException ioe) {
    ioe.printStackTrace();
    errorMes = "<BR><BR><font color=red>We are sorry, the
system could not establish connection to the NCBI PubMed
server " + "with the URL &quot;" + searchURL + "&quot;.
Please try again later.</font><BR><BR>";
    }
```

The method `getPmids()` itself looks like this:

```
  private StringBuffer getPmids(String searchURL) throws
IOException {
      BufferedReader reader = new BufferedReader(new
InputStreamReader(new
URL(searchURL).openConnection().getInputStream()));
      StringBuffer sbPmids = new StringBuffer();
      String pmid;
      String s = null;

      while ((s = reader.readLine()) != null) {
        if (pmidRE.match(s)) {
          pmid = pmidRE.getParen(1);
          sbPmids.append(pmid + ",");
        }
      }
      reader.close();
      final int length = sbPmids.length();

      if (length > 0) {
        sbPmids.delete(length - 1, length);
        return sbPmids;
      } else {
        return null;
      }
    }
```

The method:

```
  BufferedReader reader = new BufferedReader(new
InputStreamReader(new
URL(searchURL).openConnection().getInputStream()));
```

can be broken down into more readable chunks of code as follows:

```
  URLConnection urlConnection = new
URL(searchURL).openConnection();
  InputStream inputStream = urlConnection.getInputStream();
  BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
```

If no exceptions have been raised and if PMIDs have been obtained as a result of the search, we proceed to get the abstracts from the PMIDs. The method we use here is called `getAbstracts()` and returns an object of type *StringBuffer* called abstracts. The method takes a parameter called `urlAddress`, which specifies the location of the abstract based on the corresponding PMID:

```
        if (errorMes == null) {
            if (sbPmids != null) {
              String urlAddress = citationString +
URLEncoder.encode(sbPmids.toString(),
                  "UTF-8");
              StringBuffer abstracts = null;

             // 3. Retrieve the abstracts from the PubMed IDs
              try {
                abstracts = getAbstracts(urlAddress);
              } catch (IOException ioe) {
                ioe.printStackTrace();
                errorMes = "<BR><BR><font color=red>We are
sorry, the system could not retrieve the abstracts using
keyword(s) &quot;"
                    + userKeywords + "&quot; with the URL
<PRE>&quot;" + urlAddress + "&quot;<PRE></font><BR><BR>";
              }
```

The code for the method `getAbstracts()` is as follows:

```
  private StringBuffer getAbstracts(String urlAddress) throws
IOException {
      BufferedReader citationReader =
        new BufferedReader(new InputStreamReader(new
URL(urlAddress).openConnection().getInputStream()));
      StringBuffer abstracts = new StringBuffer();
```

```
    String s;
    while ((s = citationReader.readLine()) != null) {
      abstracts.append(s);
    }
    return abstracts;
  }
```

Next we get information from the matching articles corresponding to each abstract. This includes information such as the title, authors, source journal in which the article was published, and the like. An example of the MEDLINE format, which is parsed to extract this information, is shown in **Fig. 4.15**. Note the tags on the left – PMID, OWN, DP, TI, AB, AU, AD, SO. These represent respectively the PubMed ID, the owner (the organization that supplied the citation data for MEDLINE), date of publication, title, abstract, authors, address and source journal.

Fig. 4.15. The MEDLINE format

Parsing of these elements is done using the jakarta regular expression library. Let's see how we can parse the PMID from the MEDLINE record displayed above. Note that the PMID is bounded by the tags PMID and OWN as shown in the enlarged **Fig. 4.16** below.

Fig. 4.16. Parsing the PMID

We could use regular expressions to capture the PMID and other information if all the MEDLINE records had the same standard format. A few of these tags are not present wherever information is not available. For example, sometimes the abstract is not available. In such cases the AB tag is not present in the MEDLINE record which makes it a little more difficult to construct a regular expression that is generic enough for all cases. We demonstrate an alternate method that locates the position of each start and end tag and captures everything in between. We will declare the tags we will use to construct regular expressions at the beginning of the program:

```
private static final String pmidTag = "PMID- ";
private static final String pmidEndTag = "OWN - ";
private static final String titleStartTag = "TI  - ";
private static final String titleEndTag = "PG  - ";
private static final String abstractTag = "AB  - ";
private static final String abstractEndTag = "AD  - ";
private static final String fauthorStartTag = "FAU - ";
private static final String authorStartTag = "AU  - ";
private static final String authorEndTag = "LA  - ";
private static final String srcTag = "SO  -";
private static final String medlineEndTag = "</pre>";
```

We will next create code for the method that we will call `getArticleInfo() for` retrieving the information:

```
  private StringBuffer getArticleInfo(StringBuffer tmp, int
pmidStart, int endMedline) {
      StringBuffer articleTmp = new StringBuffer();
      String pmid1 = tmp.substring(pmidStart +
pmidTag.length(), tmp.indexOf(pmidEndTag));

      int titleStart = tmp.indexOf(titleStartTag);
      int titleEnd = tmp.indexOf(titleEndTag);

      if (titleEnd < 0 || titleEnd > endMedline)
        titleEnd = tmp.indexOf(abstractTag);

      int abstractStart = tmp.indexOf(abstractTag);

      if (titleEnd < 0 || titleEnd > endMedline) {
        titleEnd = tmp.indexOf(fauthorStartTag);
      }

      if (titleEnd < 0 || titleEnd > endMedline) {
        titleEnd = tmp.indexOf(fauthorStartTag);
      }
      String title = null;

      if (0 <= titleStart && titleStart < endMedline) {
        titleStart += titleStartTag.length();
        title = tmp.substring(titleStart,
titleEnd).replaceAll("(\\s+)", " ");
      }

      int end = tmp.indexOf(abstractEndTag);
      String tmpAbstractTag = abstractEndTag;

      if (end < 0 || end > endMedline) {
        end = tmp.indexOf(fauthorStartTag);
        tmpAbstractTag = fauthorStartTag;

        if (end < 0 || end > endMedline) {
          end = tmp.indexOf(authorEndTag);
          tmpAbstractTag = authorStartTag;
        }
      }

      String article = null;
      if (abstractStart + tmpAbstractTag.length() <= end) {
        article = tmp.substring(abstractStart +
tmpAbstractTag.length(), end).replaceAll("(\\s+)", " ");
      }

      int authorStart = tmp.indexOf(authorStartTag);
      String authors = null;

      if (0 <= authorStart && authorStart < endMedline) {
        authorStart += authorStartTag.length();
```

```
        int authorEnd = tmp.indexOf(authorEndTag);
        authors = tmp.substring(authorStart,
authorEnd).replaceAll(authorStartTag, ",
").replaceAll(fauthorStartTag, ", ");
      }

      int srcStart = tmp.indexOf(srcTag);
      String journal = null;
      if (0 <= srcStart && srcStart < endMedline) {
        journal = tmp.substring(srcStart + srcTag.length(),
endMedline);
      }

      // Let's create the document
      articleTmp.append("<a href=\"" + PUBMED_ARTICLE_LK +
pmid1 + "\">" + pmid1 + "</a>").append("<BR>");
      articleTmp.append("<U>Journal</u>: ");
      articleTmp.append(journal != null ? journal : "No
journal listed").append("<BR>");
      articleTmp.append("<u>Authors</u>: ");
      articleTmp.append(authors != null ? authors : "No
authors listed").append("<BR>");
      articleTmp.append("<u>Title</u>: ");
      articleTmp.append(title != null ? title : "No
title").append("<BR>");
      articleTmp.append("<u>Abstract</u>: ");
      articleTmp.append(article != null ? article : "No
article").append("<BR>");
      return articleTmp;
    }
```

The output of the second version of the PubMed servlet program that automatically parses the abstracts for each of the returned citations is shown in **Fig. 4.17**. Each of the abstracts is marked at the beginning with the PubMed ID which in turn is hyperlinked to the citation on PubMed if the user wishes to see the original record at NCBI.

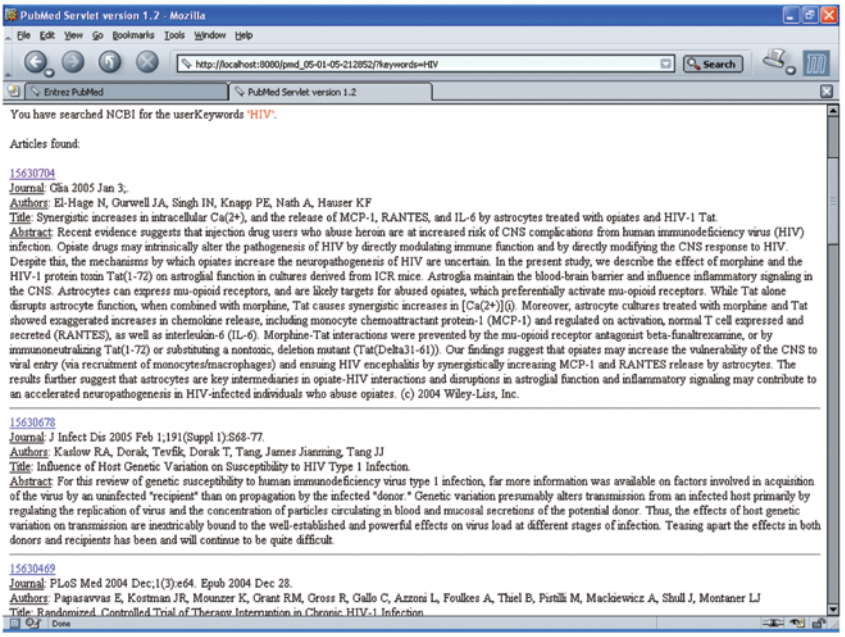

Fig. 4.17. Displaying abstracts for matching PubMed articles

The complete code for the second version of PubMed servlet (version 1.2) is shown in in **Listing 4.2**.

Listing 4.2. PubMed Servlet version 1.2

```
package org.jfb.PubMed;

import org.apache.regexp.RE;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;
import java.net.URL;
import java.net.URLEncoder;
import java.net.URLConnection;
import java.util.Properties;

public class PubMedServlet1_2 extends HttpServlet {
  private static final String TITLE = "PubMed Servlet
version 1.2";
  private static final String KEYWORDS = "keywords";
  private static final String PUBMED_ARTICLE_LK =
```

```
"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&d
b=pubmed&dopt=Abstract&list_uids=";
    private static final String citationString =

"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&d
b=PubMed&dopt=medline&list_uids=";
    private static final RE pmidRE = new RE("PMID: ([0-9]+)
\\[PubMed");

    private static final String pmidTag = "PMID- ";
    private static final String pmidEndTag = "OWN - ";
    private static final String titleStartTag = "TI  - ";
    private static final String titleEndTag = "PG  - ";
    private static final String abstractTag = "AB  - ";
    private static final String abstractEndTag = "AD - ";
    private static final String firstAuthorStartTag = "FAU -
";
    private static final String authorStartTag = "AU  - ";
    private static final String authorEndTag = "LA  - ";
    private static final String srcTag = "SO  -";
    private static final String medlineEndTag = "</pre>";

    protected void doGet(HttpServletRequest req,
HttpServletResponse res)
      throws ServletException, IOException {
      StringBuffer html = new
StringBuffer(createSearchForm());

      // 1. Extract the user-supplied keywords
      String userKeywords = getUserKeywords(req);
      if (userKeywords != null) {
        if (userKeywords.equals("")) {
          String errorMes;
          errorMes = "<BR><BR><font
color=red><b>ERROR</b><BR>Please enter keywords to
search!</font><Br><BR>";
          html.append(errorMes);
        } else {
          html.append("<BR><HR><BR>");
          html.append("You have searched NCBI PubMed with the
keywords <font color=red>'" + userKeywords + "'</font>.");

          // 2. Retrieve the PubMed IDs from the user
  // keywords
          StringBuffer sbPmids = null;   //sbpmids
          final String searchURL =
"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?dispmax=10&db=
pubmed&cmd=search&term="
            + URLEncoder.encode(userKeywords, "UTF-8");

          String errorMes = null;
```

```
          try {
  //          if (true) throw new IOException("Testing the
connection failure here!");
             sbPmids = getPmids(searchURL);
          } catch (IOException ioe) {
            ioe.printStackTrace();
            errorMes = "<BR><BR><font color=red>We are sorry,
the system could not establish connection to the NCBI PubMed
server "
              + "with the URL &quot;" + searchURL + "&quot;.
Please try again later.</font><BR><BR>";
          }

          if (errorMes == null) {
            if (sbPmids != null) {
              String urlAddress = citationString +
URLEncoder.encode(sbPmids.toString(), "UTF-8");
              StringBuffer abstracts = null;

              // 3. Retrieve the abstracts from the PubMed
  // IDs
              try {
                abstracts = getAbstracts(urlAddress);
              } catch (IOException ioe) {
                ioe.printStackTrace();
                errorMes = "<BR><BR><font color=red>We are
sorry, the system could not retrieve the abstracts using
keyword(s) &quot;"
                  + userKeywords + "&quot; with the URL
<PRE>&quot;" + urlAddress + "&quot;<PRE></font><BR><BR>";
              }

              if (errorMes == null) {
                int pmidStart = abstracts.indexOf(pmidTag);
                StringBuffer tmp = abstracts;
                html.append("<BR><br>Articles
found:<br><BR>\n");
                StringBuffer article;

                // 4. Extract information from the articles
                try {
                  while (pmidStart != -1) {
                    int endMedline =
tmp.indexOf(medlineEndTag);
                    article = getArticleInfo(tmp, pmidStart,
endMedline);
                    html.append(article);

                    tmp.delete(0, endMedline +
medlineEndTag.length());
                    pmidStart = tmp.indexOf(pmidTag);

                    if (pmidStart != -1) {
```

```
                    html.append("<HR>");
                  }
                }
              } catch (Exception e) {
                e.printStackTrace();
                errorMes = "<BR><BR><font
color=red><h1>ERROR</h1><BR>We are sorry, the system could
not retrieve the articles for PMIDs <PRE>&amp;quot;"
                  + sbPmids + "&amp;quot;<PRE></font><BR><BR>";
                html.append(errorMes);
              }
            } else {
              html.append(errorMes);
            }
          } else {
            html.append("<BR>No abstracts found!");
          }
        } else {
          html.append(errorMes);
        }
      }
    }

    appendBuildProperty(html);
    html.append("</BODY></HTML>\n");

    // 5. Print the results
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.print(html);
  }

  private String getUserKeywords(HttpServletRequest req) {
    return req.getParameter(KEYWORDS);
  }

  private StringBuffer createSearchForm() {
    StringBuffer html=new StringBuffer();
    html.append("<HTML>");
    html.append("<HEAD><TITLE>" + TITLE +
"</TITLE></HEAD><BODY>");
    html.append("<b>Java for Bioinformatics: </b>");

    html.append("<font color=red><h1>" + TITLE +
"</h1></font>");
        html.append("<BR><B>Please enter a term to search
on NCBI PubMed: </B><BR><BR>\n");

    html.append("<FORM METHOD=GET>\n");
    html.append("<INPUT TYPE=TEXT NAME=" + KEYWORDS +
"><BR><BR>\n");
    html.append("<INPUT TYPE=SUBMIT VALUE=\"Search
PubMed\"><BR>\n");
```

```
        html.append("</FORM>\n");
        return html;
    }

    private StringBuffer getPmids(String searchURL) throws
IOException {
        URLConnection urlConnection = new
URL(searchURL).openConnection();
        InputStream inputStream =
urlConnection.getInputStream();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));

        StringBuffer sbPmids = new StringBuffer();
        String pmid;
        String s = null;

        while ((s = reader.readLine()) != null) {
          if (pmidRE.match(s)) {
            pmid = pmidRE.getParen(1);
            sbPmids.append(pmid + ",");
          }
        }
        reader.close();
        final int length = sbPmids.length();

        if (length > 0) {
          sbPmids.delete(length - 1, length);
          return sbPmids;
        } else {
          return null;
        }
    }

    private StringBuffer getAbstracts(String urlAddress)
throws IOException {
        BufferedReader citationReader =
          new BufferedReader(new InputStreamReader(new
URL(urlAddress).openConnection().getInputStream()));
        StringBuffer abstracts = new StringBuffer();

        String s;
        while ((s = citationReader.readLine()) != null) {
          abstracts.append(s);
        }
        return abstracts;
    }

    private StringBuffer getArticleInfo(StringBuffer tmp, int
pmidStart, int endMedline) {
        StringBuffer articleTmp = new StringBuffer();
        String pmid1 = tmp.substring(pmidStart +
pmidTag.length(), tmp.indexOf(pmidEndTag));
```

```
      int titleStart = tmp.indexOf(titleStartTag);
      int titleEnd = tmp.indexOf(titleEndTag);

      if (titleEnd < 0 || titleEnd > endMedline)
        titleEnd = tmp.indexOf(abstractTag);

      int abstractStart = tmp.indexOf(abstractTag);

      if (titleEnd < 0 || titleEnd > endMedline) {
        titleEnd = tmp.indexOf(firstAuthorStartTag);
      }
      String title = null;

      if (0 <= titleStart && titleStart < endMedline) {
        titleStart += titleStartTag.length();
        title = tmp.substring(titleStart,
titleEnd).replaceAll("(\\s+)", " ");
      }

      int end = tmp.indexOf(abstractEndTag);
      String tmpAbstractTag = abstractEndTag;

      if (end < 0 || end > endMedline) {
        end = tmp.indexOf(firstAuthorStartTag);
        tmpAbstractTag = firstAuthorStartTag;

        if (end < 0 || end > endMedline) {
          end = tmp.indexOf(authorEndTag);
          tmpAbstractTag = authorStartTag;
        }
      }

      String article = null;
      if (abstractStart + tmpAbstractTag.length() <= end) {
        article = tmp.substring(abstractStart +
tmpAbstractTag.length(), end).replaceAll("(\\s+)", " ");
      }

      int authorStart = tmp.indexOf(authorStartTag);
      String authors = null;

      if (0 <= authorStart && authorStart < endMedline) {
        authorStart += authorStartTag.length();
        int authorEnd = tmp.indexOf(authorEndTag);
        authors = tmp.substring(authorStart,
authorEnd).replaceAll(authorStartTag, ",
").replaceAll(firstAuthorStartTag, ", ");
      }

      int srcStart = tmp.indexOf(srcTag);
      String journal = null;
      if (0 <= srcStart && srcStart < endMedline) {
```

```
        journal = tmp.substring(srcStart + srcTag.length(),
endMedline);
      }

      // Let's create the document
      articleTmp.append("<a href=\"" + PUBMED_ARTICLE_LK +
pmid1 + "\">" + pmid1 + "</a>").append("<BR>");
      articleTmp.append("<U>Journal</u>: ");
      articleTmp.append(journal != null ? journal : "No
journal listed").append("<BR>");
      articleTmp.append("<u>Authors</u>: ");
      articleTmp.append(authors != null ? authors : "No
authors listed").append("<BR>");
      articleTmp.append("<u>Title</u>: ");
      articleTmp.append(title != null ? title : "No
title").append("<BR>");
      articleTmp.append("<u>Abstract</u>: ");
      articleTmp.append(article != null ? article : "No
article").append("<BR>");
      return articleTmp;
    }

    private void appendBuildProperty(StringBuffer html) {
      Properties buildInfo = null;

      try {
        buildInfo = new Properties();
        InputStream buildStream =
getClass().getClassLoader().getResourceAsStream("/build-
info.txt");
        buildInfo.load(buildStream);
      } catch (Throwable e) {
        e.printStackTrace();
      }

      if (buildInfo != null) {
        html.append("<BR><HR><font color=grey>Build #");
        html.append(buildInfo.getProperty("buildNumber"));
        html.append("</font>\n");
      }
    }

    public static void main(String[] args) throws Exception {
      new PubMedServlet1_2();
    }
  }
```

Highlighting Search Terms in Retrieved Abstracts

In version 1.3 of the PubMed servlet, we will enhance the usefulness of the search results by highlighting the search terms in the retrieved

abstracts. One way to do this is to convert the search terms and the abstract into lower case, locate the matches and then highlight the terms in the abstract. In this method, we lose the case of the words in the original abstract (because we converted that into lower case). To fix this, we could find the exact location of the match and the length of the match and use the original abstract to highlight the matching term(s).

Another way is to use the `equalsIgnoreCase()` method which compares strings irrespective of case. For example, the following code will find a match to the term "HIV" in text even if it contains HIV in different forms such as hiv, Hiv, HIv, hIV, etc.

```
if (word.equalsIgnoreCase( "HIV" ) ) {
  //code for highlighting matching terms;
}
```

To use this method, we have to first create an array of words in the abstract and test if any of the individual words match the search term. However, there are limitations to this method also. The `equalsIgnoreCase()` method searches for exact matches and will not find words containing punctuation marks and other characters. If, for example, HIV-1 is found at the end of a sentence, the array element will be "HIV." (with a period) and "HIV" is not equal to "HIV.". To fix this we need to get rid of all such punctuation marks and other special characters.

An easier method to circumvent these issues is described below. In this method, we iterate over the text in the abstract highlighting each term as it is found. The regular expression itself is of the type:

```
(a|A) (b|B) (c|C)...
```

which will match any word irrespective of case. Surrounding such expression in parentheses allows us to extract specific sub-strings from a string based on a specified pattern. This is implemented in code as follows:

```
StringBuffer sb = new StringBuffer("(");
for (char c : chars) {
    char charUp = Character.toUpperCase(c);
    char charLo = Character.toLowerCase(c);

sb.append("(").append(charLo).append("|").append(charUp).appe
nd(")");
}
sb.append(")");
```

```
  final String regex = "^" + sb.toString() + "|[^a-zA-Z]" +
sb.toString();
```

We will not only highlight the search term in the abstracts, we will also color them differently for better visibility and readability. To do this, we need to declare an array called COLOR of color elements to store the selection of colors we wish to use:

```
  private static final String[] COLOR = new String[]{"blue",
"#98cc02", "purple", "red", "#f7dc88"};
```

For each of the characters in the search term, a regular expression of the type indicated above (with both lower and upper case forms) is created. Next when the term is found in the article text, it is highlighted using a different color for each matching term.

```
  highlightedText = re.subst(highlightedText, "\\\\<b><font
style=\"\\\\+2\" color=\"" + COLOR[i] + "\">$0</font></b>",
                        RE.REPLACE_BACKREFERENCES);
          }
```

The complete code for the method which we will call `highlight()` is as follows:

```
  private String highlight(String articleText, String[]
terms) {
          String highlightedText = new String(articleText);
          for (int i = 0; i < terms.length; i++) {
              final String term = terms[i];
              final char[] chars = term.toCharArray();

  // Here we are creating the regular expression to find any
// word irrespective of case.
              StringBuffer sb = new StringBuffer("(");
              for (char c : chars) {
                char charUp = Character.toUpperCase(c);
                char charLo = Character.toLowerCase(c);

sb.append("(").append(charLo).append("|").append(charUp).appe
nd(")");
              }
              sb.append(")");
              final String regex = "^" + sb.toString() +
"|[^a-zA-Z]" + sb.toString();

              // Replace the text by a HTML FONT tag that
  // wraps the term found
              RE re = new RE(regex);
```

```
                highlightedText = re.subst(highlightedText,
"\\\\<b><font style=\"\\\\+2\" color=\"" + COLOR[i] +
"\">$0</font></b>", RE.REPLACE_BACKREFERENCES);
            }
            return highlightedText;
        }
```

The regular expression for highlighting matched text with colored text is constructed using the Jakarta regular expression library. In particular we are using the *subst* method (short for substring), which is defined as follows:

```
re.subst(string1, string2, rules)
```

where,

`string1`: the *String* to make the substitution in
`string2`: *String* to substitute into `string1`
`rules`: rules that define how substitutions are to be done in `string1`

To refer to the contents of a parenthesized expression within a regular expression, we use what are known as *'backreferences'*. The first backreference in a regular expression is denoted by \1, the second by \2 and so on.

The rules are set as follows:

REPLACE_FIRSTONLY: replace only the first occurrence of the regular expression in string1

REPLACE_ALL: replace all occurrences of the regular expression in string1

REPLACE_BACKREFERENCES: all backreferences will be processed, which in this case means that all matched patterns within the article text will be replaced with string2

In our case,

`string1` = highlightedText
`string2` = "\\\\$0"
and

```
rules = RE.REPLACE_BACKREFERENCES
```

The extra backslashes in `string2` are *escape characters*. Note that the expression "`$0`" represents the whole match, which in this case, represent the search term(s). The output of PubMed servlet version 1.3 obtained from an ANDed search of the terms HIV AND AIDS is shown in **Fig. 4.18**.

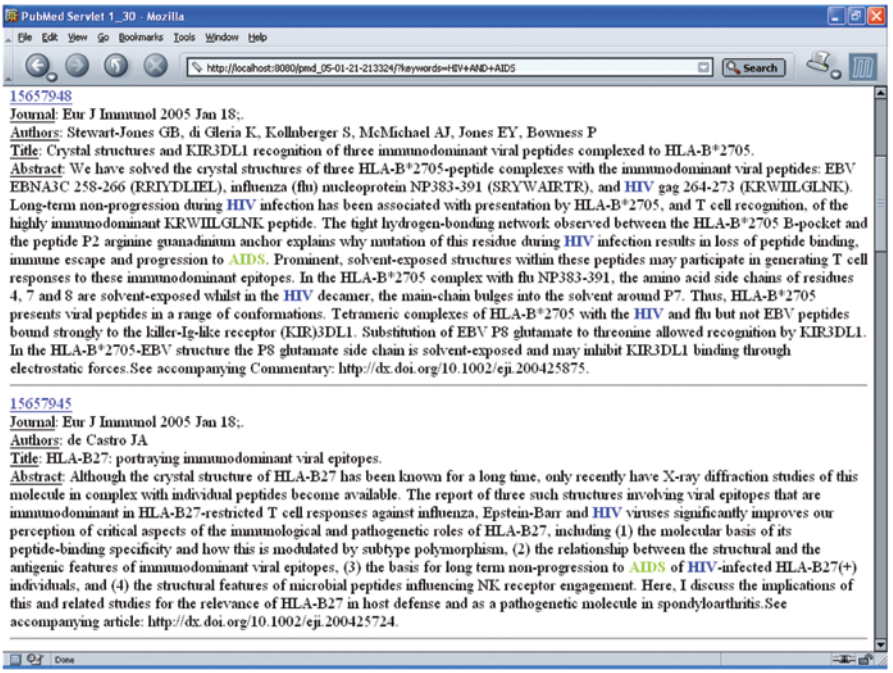

Fig. 4.18. Highlighting search terms in PubMed abstracts

As the output shows, both keywords have been highlighted (blue and green respectively, as specified in the array of HTML colors).

The complete code for PubMed servlet version 1.3 is shown in **Listing 4.3**.

Listing 4.3. PubMed Servlet version 1.3

```
package org.jfb.PubMed;

import org.apache.regexp.RE;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
  import javax.servlet.http.HttpServletResponse;
  import java.io.*;
  import java.net.URL;
  import java.net.URLEncoder;
  import java.util.Properties;

  public class PubMedServlet1_3 extends HttpServlet {
      private static final String TITLE = "PubMed Servlet
1_30";
      private static final String KEYWORDS = "keywords";
      private static final String PUBMED_ARTICLE_LK =

"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&d
b=pubmed&dopt=Abstract&list_uids=";
      private static final String citString =

"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&d
b=PubMed&dopt=medline&list_uids=";
      private static final RE pmidRE = new RE("PMID: ([0-9]+)
\\[PubMed");

      private static final String pmidTag = "PMID- ";
      private static final String pmidEndTag = "OWN - ";
      private static final String titleStartTag = "TI  - ";
      private static final String titleEndTag = "PG  - ";
      private static final String abstractTag = "AB  - ";
      private static final String abstractEndTag = "AD  - ";
      private static final String firstAuthorStartTag = "FAU
- ";
      private static final String authorStartTag = "AU  - ";
      private static final String authorEndTag = "LA  - ";
      private static final String srcTag = "SO  -";
      private static final String medlineEndTag = "</pre>";
      private static final String[] COLOR = new
String[]{"blue", "#98cc02", "purple", "red", "#f7dc88"};
      private String[] params;

      protected void doGet(HttpServletRequest req,
HttpServletResponse res) throws ServletException, IOException
{
          StringBuffer html = new StringBuffer();

          // 1. Retrieve the user supplied keywords
          printHeader(html);
          String userKeywords = req.getParameter(KEYWORDS);

          if (userKeywords != null) {
              params =
userKeywords.replaceAll("\\s*(\\+|((a|A)(N|n)(D|d))|((o|O)(r|
R)))\\s*", " ").split(" ");
              html.append("<BR><HR><BR>");
              html.append("You have searched NCBI for the
userKeywords '"
```

```
                        + highlight(userKeywords, this.params)
+ "'.");

                // 2. Retrieve the PubMed IDs from abstracts
  // matching user supplied keywords.
                StringBuffer sbPmids = null;
                final String spec =
"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?dispmax=10&db=
pubmed&cmd=search&term="
                        + URLEncoder.encode(userKeywords, "UTF-
8");

                String errorMes = null;
                System.out.println("spec = " + spec);
                try {
                    sbPmids = getPmids(spec);
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                    errorMes = "<BR><BR><font color=red>We are
sorry, the system could not retrieve the PubMed IDs using
keyword(s) &quot;"
                            + userKeywords + "&quot; with the
URL <PRE>&quot;" + spec + "&quot;<PRE></font><BR><BR>";
                }

                if (errorMes == null) {
                    if (sbPmids != null) {
                        String urlAddress = citString +
URLEncoder.encode(sbPmids.toString(), "UTF-8");
                        StringBuffer abstracts = null;

                        // 3. Retrieve abstracts corresponding
// to the PubMed IDs
                        try {
                            abstracts =
getAbstracts(urlAddress);
                        } catch (IOException ioe) {
                            ioe.printStackTrace();
                            errorMes = "<BR><BR><font
color=red>We are sorry, the system could not retrieve the
abstracts using keyword(s) &quot;"
                                    + userKeywords + "&quot;
with the URL <PRE>&quot;" + urlAddress +
"&quot;<PRE></font><BR><BR>";
                        }

                        if (errorMes == null) {
                            int pmidStart =
abstracts.indexOf(pmidTag);
                            StringBuffer tmp = abstracts;
                            html.append("<BR><br>Articles
found:<br><BR>\n");
                            StringBuffer article;
```

```
                            // 4. Formatt the articles
                            try {
                                while (pmidStart != -1) {
                                    int endMedline =
tmp.indexOf(medlineEndTag);
                                    article = getArticle(tmp,
pmidStart, endMedline);
                                    html.append(article);

                                    tmp.delete(0, endMedline +
medlineEndTag.length());
                                    pmidStart =
tmp.indexOf(pmidTag);

                                    if (pmidStart != -1) {
                                        html.append("<HR>");
                                    }
                                }
                            } catch (Exception e) {
                                e.printStackTrace();
                                errorMes = "<BR><BR><font
color=red><h1>ERROR</h1><BR>We are sorry, the system could
not retrieve the articles for PMIDs <PRE>&amp;quot;"
                                        + sbPmids +
"&amp;quot;<PRE></font><BR><BR>";
                                html.append(errorMes);
                            }
                        } else {
                            html.append(errorMes);
                        }
                    } else {
                        html.append("<BR>No abstracts found!");
                    }
                } else {
                    html.append(errorMes);
                }
            }

            appendBuildProperty(html);
            html.append("</BODY></HTML>\n");

            // 5. Print the results
            res.setContentType("text/html");
            PrintWriter out = res.getWriter();
            out.print(html);
        }

        private void printHeader(StringBuffer html) {
            html.append("<HTML>");
            html.append("<HEAD><TITLE>" + TITLE +
"</TITLE></HEAD><BODY>\n");
```

```
            html.append("<font color=red><h1>" + TITLE +
"</h1></font>\n");
            html.append("<B>Please enter a userKeywords to
search on NCBI:</B><BR><BR>\n");
            html.append("<FORM METHOD=GET>\n");
            html.append("<INPUT TYPE=TEXT NAME=" + KEYWORDS +
"><BR><BR>\n");
            html.append("<INPUT TYPE=SUBMIT VALUE=\"Search
PubMed\"><BR>\n");
            html.append("</FORM>\n");
        }

        private StringBuffer getPmids(String spec) throws
IOException {
            BufferedReader reader = new BufferedReader(new
InputStreamReader(new
URL(spec).openConnection().getInputStream()));
            StringBuffer sbPmids = new StringBuffer();
            String pmid;
            String s = null;

            while ((s = reader.readLine()) != null) {
                if (pmidRE.match(s)) {
                    pmid = pmidRE.getParen(1);
                    sbPmids.append(pmid + ",");
                }
            }
            reader.close();
            final int length = sbPmids.length();

            if (length > 0) {
                sbPmids.delete(length - 1, length);
                return sbPmids;
            } else {
                return null;
            }
        }

        private StringBuffer getAbstracts(String urlAddress)
throws IOException {
            BufferedReader citReader =
                    new BufferedReader(new
InputStreamReader(new
URL(urlAddress).openConnection().getInputStream()));
            StringBuffer absSb = new StringBuffer();

            String s;
            while ((s = citReader.readLine()) != null) {
                absSb.append(s);
            }
            return absSb;
        }
```

```
      private StringBuffer getArticle(StringBuffer tmp, int
pmidStart, int endMedline) {
          StringBuffer articleTmp = new StringBuffer();
          String pmid1 = tmp.substring(pmidStart +
pmidTag.length(), tmp.indexOf(pmidEndTag));

          int titleStart = tmp.indexOf(titleStartTag);
          int titleEnd = tmp.indexOf(titleEndTag);

          if (titleEnd < 0 || titleEnd > endMedline)
              titleEnd = tmp.indexOf(abstractTag);

          int abstractStart = tmp.indexOf(abstractTag);

          if (titleEnd < 0 || titleEnd > endMedline) {
              titleEnd = tmp.indexOf(firstAuthorStartTag);
          }

          if (titleEnd < 0 || titleEnd > endMedline) {
              titleEnd = tmp.indexOf(firstAuthorStartTag);
          }
          String title = null;

          if (0 <= titleStart && titleStart < endMedline) {
              titleStart += titleStartTag.length();
              title = tmp.substring(titleStart,
titleEnd).replaceAll("(\\s+)", " ");
          }

          int end = tmp.indexOf(abstractEndTag);
          String tmpAbstractTag = abstractEndTag;

          if (end < 0 || end > endMedline) {
              end = tmp.indexOf(firstAuthorStartTag);
              tmpAbstractTag = firstAuthorStartTag;

              if (end < 0 || end > endMedline) {
                  end = tmp.indexOf(authorEndTag);
                  tmpAbstractTag = authorStartTag;
              }
          }

          String article = null;
          if (abstractStart + tmpAbstractTag.length() <= end)
{
              article = tmp.substring(abstractStart +
tmpAbstractTag.length(), end).replaceAll("(\\s+)", " ");
          }

          int authorStart = tmp.indexOf(authorStartTag);
          String authors = null;

          if (0 <= authorStart && authorStart < endMedline) {
```

```
                authorStart += authorStartTag.length();
                int authorEnd = tmp.indexOf(authorEndTag);
                authors = tmp.substring(authorStart,
authorEnd).replaceAll(authorStartTag, ",
").replaceAll(firstAuthorStartTag, ", ");
            }

            int srcStart = tmp.indexOf(srcTag);
            String journal = null;
            if (0 <= srcStart && srcStart < endMedline) {
                journal = tmp.substring(srcStart +
srcTag.length(), endMedline);
            }

            // Create the output
            articleTmp.append("<a href=\"" + PUBMED_ARTICLE_LK
+ pmid1 + "\">" + pmid1 + "</a>").append("<BR>");
            articleTmp.append("<U>Journal</u>: ");
            articleTmp.append(journal != null ? journal : "No
journal listed").append("<BR>");
            articleTmp.append("<u>Authors</u>: ");
            articleTmp.append(authors != null ? authors : "No
authors listed").append("<BR>");
            articleTmp.append("<u>Title</u>: ");
            articleTmp.append(title != null ? highlight(title,
params) : "No title").append("<BR>");
            articleTmp.append("<u>Abstract</u>: ");
            articleTmp.append(article != null ?
highlight(article, params) : "No article").append("<BR>");
            return articleTmp;
        }

        private String highlight(String articleText, String[]
terms) {
            String highlightedText = new String(articleText);
            for (int i = 0; i < terms.length; i++) {
                final String term = terms[i];
                final char[] chars = term.toCharArray();

    // Create the regular expression to find search terms
    // irrespective of case

                StringBuffer sb = new StringBuffer("(");
                for (char c : chars) {
                  char charUp = Character.toUpperCase(c);
                  char charLo = Character.toLowerCase(c);

sb.append("(").append(charLo).append("|").append(charUp).appe
nd(")");
                }
                sb.append(")");
                final String regex = "^" + sb.toString() +
"|[^a-zA-Z]" + sb.toString();
```

```
                // Replace the text by a HTML FONT tag
  // that wraps the term found
                RE re = new RE(regex);
                highlightedText = re.subst(highlightedText,
"\\\\<b><font style=\"\\\\+2\" color=\"" + COLOR[i] +
"\">$0</font></b>",
                        RE.REPLACE_BACKREFERENCES);
            }
            return highlightedText;
        }
```

In this Chapter, we have attempted to demonstrate how web applications can be created using the J2EE JSP and servlets technology based on a literature search and retrieval service that is indispensable for today's fast paced scientific research environment. In particular, we created a web application that provides the same powerful search capabilities provided by the NCBI PubMed server but further enhanced it by displaying the abstracts for each of the matching articles right up front and highlighting the search terms in the abstract. The rationale behind this strategy was that researchers may find it difficult to recognize the relevance of an article to their area of research simply by looking at the article title. If the abstract was displayed and the search terms were highlight and color coded, it becomes much easier to understand the context in which the abstract is relevant vis-à-vis the input search terms. This design saves the researcher a few extra clicks and makes data more readable and useful.

> Note: This Chapter uses resources referred to in the Appendix: Setting up Apache ant and Apache Tomcat.

Summary

The ability to query and mine the rich scientists datasets in PubMed is a powerful way to further experimental science using a hypothesis driven research methodology where researchers build on scientific findings reported by scores of researchers around the world. In this Chapter, we have demonstrated how to create a web application with Java *Servlet/JSP* technology to access PubMed data and how to enhance the functionality provided by the resource. Processing and presentation of biomedical data in ways that provide additional benefit for the researcher is a fundamental contribution of information technologies and it is hoped that this Chapter has illustrated a small example of how this can be accomplished.

Questions and Exercises

1. Visit the NCBI PubMed website and become familiar with the service. Try out searches with different keywords and view the results using the various available Display (Brief, Abstract, Citation, XML, etc.), Sort by (Pub Date, First Author, Last Author, etc.) and Limits (Dates, Type of Article, etc.) options. Think of ways you can enhance the capabilities of the service from the user's point-of-view.

2. PubMed abstracts are a powerful source of data on protein-protein interaction networks. For example, two or more proteins mentioned in the same sentence within an abstract most likely interact with or are related to one another in some fashion. Enhance the PubMed web application we created in the Chapter by:
   a. highlighting gene/protein names mentioned in the abstract
   b. hyperlinking protein names to an appropriate annotation resource or database on the web

One such solution can be based on the use of gene symbols defined by the HUGO Gene Nomenclature Committee (HGNC). According to HGNC convention, human gene symbols are designated by upper-case Latin letters or by a combination of upper-case letters and Arabic numerals, with some exceptions. For example, the Approved Gene Symbol for the breast cancer 1, early onset gene is BRCA1.

For the second part of the exercise, the NCBI Entrez Gene resource can be used as an annotation resource. The link to the BRCA1 gene on Entrez Gene, for example, is identified by the following URL:

http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=gene&cmd=Retrieve&dopt=full_report&list_uids=672

3. Enhance the user interface of the web application to include the capability to:
   a. save selected abstracts on your local machine
   b. filter articles by special criteria, for example, limit journals by name (Science, Nature, etc.)

Additional Resources

- The Apache Software Foundation - http://tomcat.apache.org
- The Apache Jakarta Project - http://jakarta.apache.org/regexp/
- The Apache Ant Project - http://ant.apache.org/
- Entrez - http://www.ncbi.nlm.nih.gov/Database/index.html
- HUGO Gene Nomenclature Committee - http://www.gene.ucl.ac.uk/nomenclature/
- Java Servlet API Specification 2.2 - http://java.sun.com/products/servlet/download.html
- JavaServer Pages[tm] Technology - White Paper - http://java.sun.com/products/jsp/whitepaper.html
- The Java Servlet API White Paper - http://java.sun.com/products/servlet/whitepaper.html
- Java Servlet Technology - http://java.sun.com/products/servlet/index.jsp
- PubMed Help website - http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=helppubmed.chapter.pubmedhelp
- RFC 2616 - http://www.w3.org/Protocols/rfc2616/rfc2616.html
- RFC 3875 - http://www.rfc-archive.org/getrfc.php?rfc=3875

Selected Reading

The HUGO Gene Nomenclature Database, 2006 updates. Eyre TA, Ducluzeau F, Sneddon TP, Povey S, Bruford EA and Lush MJ. Nucleic Acids Res. 2006 Jan 1;34(Database issue):D319-21.

Guidelines for human gene nomenclature (1997). HUGO Nomenclature Committee. White JA, McAlpine PJ, Antonarakis S, Cann H, Eppig JT, Frazer K, Frezal J, Lancet D, Nahmias J, Pearson P, Peters J, Scott A, Scott H, Spurr N, Talbot C Jr, Povey S. Genomics. 1997 Oct 15;45(2):468-71.