Chapter III

Running BLAST using SwingBlast

Introduction

In the last Chapter, we created the basic framework application called `SwingBlast` Version 1.3 using Swing libraries to manipulate user defined nucleotide and protein sequences and prepare them for BLAST searches. In this Chapter, we will add functionality to the application that enables users to download sequences automatically from NCBI GenBank, submit sequences for multiple simultaneous BLAST analyses, and save and view BLAST results. To begin with, we will demonstrate how to use NCBI's *QBlast* package to perform BLAST searches. We will then create an application called JQBlast to demonstrate how to use theQBlast package to run BLAST searches.

The NCBI QBLAST Package

NCBI provides a standardized API called URLAPI to formulate and dispatch direct HTTP-encoded requests to the NCBI QBlast system. The URLAPI provides a URL and a mechanism to set parameters that allows users to send sequences for BLAST searches.

NCBI QBlast works through 4 steps:

1. The user provides BLAST parameters through a URL using the HTTP POST method

2. The QBlast service returns a Request Identifier (RID) and a Request Time of Execution (RTOE, measured in seconds) for the search, which provide respectively, a unique identifier for the search operation and an estimate of the time required to complete the search

3. The user queries the QBlast service with the RID through HTTP GET method

4. The server sends back the result with a status value that indicates the progress of the BLAST request

Users of the QBlast service should adhere to the guidelines provided by NCBI when submitting large batch searches. In general, searches should be performed in a sequential manner after receiving the RID and RTOE for each submission. NCBI specifies that each request be submitted after a pause of no less than 3 seconds to check on the status of the request using the RID. Failure to do so may overload the server and force NCBI to block offending users from further use of the service.

Strategy for Creating a QBlast Based System

The design of the NCBI QBlast service as described above stipulates the need for a client application that performs the following operations:

1. Send search requests made by the user and check the status of requests periodically

2. Perform the appropriate action based on the nature of the status value that gets returned

QBlast may return one of three types of status values: "READY" meaing that the search was completed successfully, "WAITING" meaning that the search has not been completed and "UNKNOWN" meaning that an error has been encountered during the BLAST submission and/or search process. In UML terms, the user and the client application are actors that interact with the QBlast system. The UML diagram below (**Fig. 3.1**) depicts the use cases that encapsulate the basic functionality that is desired of the system that we wish to create:

1. User submits query sequence to the QBlast service

2. Application queries status of the BLAST search with a unique RID

3. Applcation returns results approriate to the status value

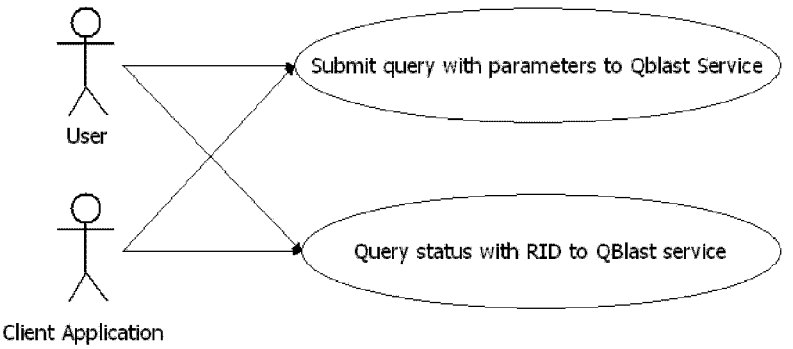

Fig. 3.1. Use Cases for the QBlast service

In terms of the architecture of the application, we will provide a class that will wrap the NCBI URLAPI into Java API that can be reused in other applications. To fulfill these use cases, we will design the QBlast service to implement 2 methods: `submitQuery` and `queryStatus` (**Fig. 3.2**).

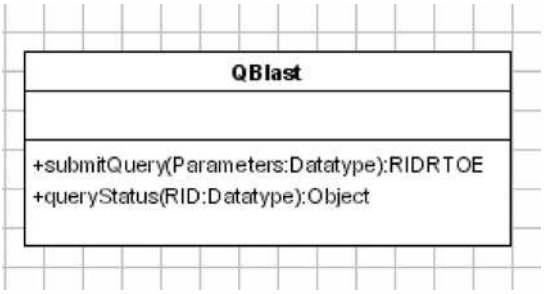

Fig. 3.2. Class QBlast

The `QBlast` class is our interface to the real NCBI URLAPI. From the application point of view, it is totally transparent and is designed to be so in order to accommodate and simplify future changes to the API (or, if there is a need to adopt an entirely different API). This design ensures that the framework we create remains usable even if the underlying API requires changes. The `submitQuery()` method takes the BLAST

parameters (specified through the `QBlastParameter` object) and returns an object of type `RequestIdentifier`. The parameters needed to run the BLAST search would be obtained from the user through the `SwingBlast` GUI we created in Chapter 1. The `RequestIdentifier` is returned by the QBlast service in response to the submitted request and contains the RID and the RTOE for a specific search.

For the `queryStatus` method similarly, we will need 2 objects: `RequestIdentifier` and `QBlastResult`. A UML diagram with these considerations in mind is shown in **Fig. 3.3.**

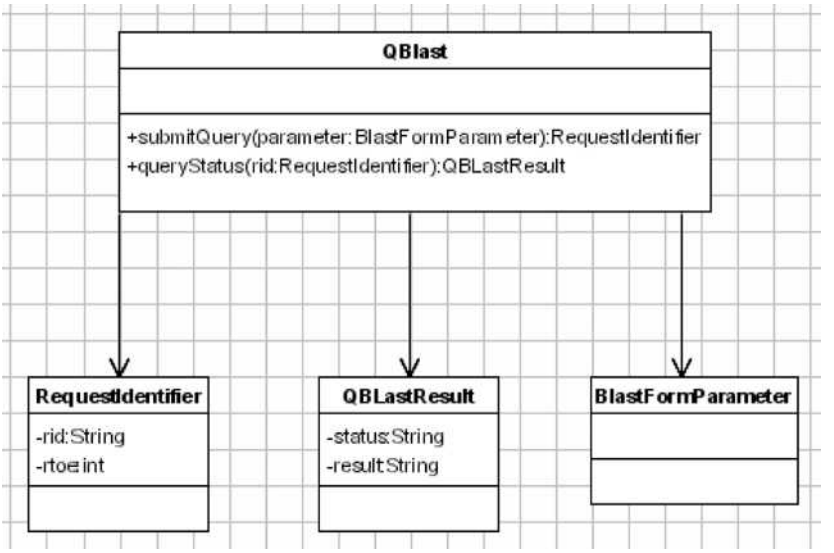

Fig. 3.3. UML class diagram showing the QBlast architecture

Designing the BLAST API

We will design our BLAST API to consist of 3 classes:

- `Blast`
- `BlastManager`
- `BlastException`

We will define `Blast` as an *abstract class*, which means that it represents an abstract concept, and therefore cannot be instantiated, but can only be subclassed. An abstract class is declared using the keyword *abstract* before the class keyword in the class declaration. In this case, for example, we would declare the *Blast* class as shown below:

```
abstract class blast { … }
```

We'll describe this class in detail later in the Chapter. The *BlastManager* class provides a mechanism to get an instance of the abstract class *Blast* without having to worry about how to create the instance by calling the *static* method (that we had earlier explained in Chapter 2):

```
Blast blast = BlastManager.createBlast();
```

The `BlastException` class provides a mechanism for handling exceptions thrown by any implementation when a failure or error occurs. The `RequestIdentifier` class is a Java class, which provides what are known as *setter*, and *getter* methods that provide information about the request submitted to the Blast service. What are *setter* and *getter* methods? In a class definition, private fields can be encapsulated so that the data structure used can be changed at will without compromising the rest of the code that uses that class. When the data structure is hidden, the way to provide access to and/or modify the fields is through setter and getter methods. For example, a class that has a field called `result` will provide a *setter* method called `setResult` and a *getter* method call `getResult`. The `RequestIdentifier` class uses these methods as described above. The structure of the application designed so far is shown in **Fig. 3.4**.

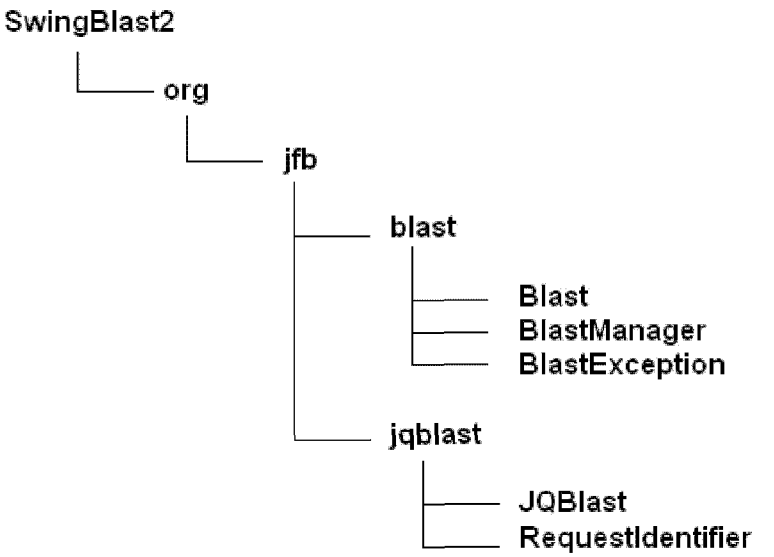

Fig. 3.4. Structure of the SwingBlast application

Description of Blast Classes

The `Blast` class extends the *Observable* class, which represents an observable object, an instance of which can be observed for any changes that occur to the object. When an observable instance changes (that is, when an object that is being observed changes), the `notifyObservers` method is called and causes the observer to be notified of the change by a call to the observer's update method. In this case, we want to observe the `Blast` class for changes that occur during the process of submitting the request and waiting for the result to be returned. We can then notify the observers of the progress of the search, as well as when the results are ready or if an error occurs.

The `Blast` class contains 2 abstract methods:

```
submitQuery()
```

and,

```
requestResult()
```

that respectively take one parameter each: a Java data type called *Map* for the BLAST parameters, which stores sets of elements in the form of key-value pairs, and the identifier for the identifier that is uniquely associated with each BLAST search. Both methods return an object of the respective type and throw an exception if an error occurs. The `Blast` class is defined in **Listing 3.1**.

Listing 3.1. The Blast class

```
package org.jfb.blast;

import java.util.HashMap;
import java.util.Observable;

public abstract class Blast extends Observable {
public abstract Object submitQuery(Map parameters)
throws BlastException;

  public abstract Object requestResult(Object identifier)
throws BlastException;
}
```

The way to initialize the `BlastManager` is to provide the full class name of the implementation through a *JVM system property* called '`blast.driver`'. An example of how to provide our BLAST implementation called `org.jfb.jqblast.JQBlast` to the BlastManager via a JVM system property is shown below:

```
Java -Dblast.driver=org.jfb.jqblast.JQBlast ...
```

The class `JQBlast` must be declared in the *Java classpath* to be able to be found by the *Java classloader*. The Java classloader is responsible for loading a Java class when it is needed. The `BlastManager` class is described in **Listing 3.2**.

Listing 3.2. The BlastManager class

```
package org.jfb.blast;

public class BlastManager {
    private static String blastClass = null;
    private static boolean initialized = false;

    public static void register(Blast blast) {
        blastClass = blast.getClass().getName();
    }
```

```
    private static void loadInitialDrivers() {
        final           String          driver          =
System.getProperty("blast.driver");
        if (driver == null)
            return;

        try {
            System.out.println("BlastManager.Initialize:
loading " + driver);
            Class.forName(driver);
        } catch (Exception e) {
            System.out.println("BlastManager.Initialize:
load failed: " + e);
        }
    }

    public static Blast createBlast() throws BlastException
{
        if (!initialized) {
            initialized = true;
            loadInitialDrivers();
        }
        if (blastClass == null)
            throw new BlastException("There is no driver
configured! "
                    +  "Please  use  blast.driver  Java
property or Class.forName to load the driver class.");
        try {
            // In a multi thread environment we need to
make sure // that the class is loaded.
            final     Class     aClass     =     (Class)
Class.forName(blastClass,                          true,
Thread.currentThread().getContextClassLoader());
            return    (Blast)    aClass.getConstructor(new
Class[]{}).newInstance(new Object[]{});
        } catch (Exception e) {
            throw new BlastException(e);
        }
    }
}
```

The purpose of the register() method is to inform the BlastManager which Blast implementation we want to use. This is done as follows:

```
    public static void register(Blast blast) {

        blastClass = blast.getClass().getName();

    }
```

Here, `blast.getClass()` returns an instance of class `java.lang.Class`. `blastClass` is an instance of `java.lang.Class` and `blastClass.getName()` will return the real class name which, in this case would be `org.jfb.jqblast.JQBlast`.

Let's look at the `loadInitialDrivers` method below:

```
  private static void loadInitialDrivers() {
          final          String          driver          =
System.getProperty("blast.driver");
          if (driver == null)
              return;

          try {
              System.out.println("BlastManager.Initialize:
loading " + driver);
              Class.forName(driver);
          } catch (Exception e) {
              System.out.println("BlastManager.Initialize:
load failed: " + e);
          }
      }
```

When the `loadInitialDrivers` method is called, it gets the property `blast.driver` from the system and if it is not null, calls the `Class.forName()` method. At that point, `BlastManager` knows that a `Blast` driver is registered and available, otherwise an exception is thrown with an error message. Finally, the `BlastException` class handles any exceptions that arise during the BLAST search (**Listing 3.3**).

Listing 3.3. The BlastException class

```
  package org.jfb.blast;

  public class BlastException extends Exception {
      public BlastException() {
      }

      public BlastException(String message) {
          super(message);
      }

      public BlastException(String message, Throwable cause)
{
          super(message, cause);
      }

      public BlastException(Throwable cause) {
```

```
            super(cause);
        }
    }
```

Implementing JQBlast

We will now build the `JQBlast` application that allows users to send multiple simultaneously BLAST queries using the classes we described above. To implement the NCBI QBlast package, we just need to extend the `Blast` class and provide an implementation of the methods as described above. We will call the instance of the `Blast` class `JQBlast` as shown below:

```
public class JQBlast extends Blast {
    //implement Blast methods
}
```

We will create a file called `QBlast.java` to implement this code. It is up to the developer of a `Blast` implementation to provide the code for those methods. The developer must also register the `Blast` class to the `BlastManager` class using a *static* statement that will be executed after loading the class. A *static* statement is a piece of code that starts with the Java keyword *static* and is followed by curly brackets (which, in this case, holds the code that loads the Blast implementation called `org.jfb.jqblast.JQBlast`). It is executed after the class is loaded in the JVM:

```
public class JQBlast extends Blast {
static {
        System.out.println("Registering " + JQBlast.class);
        BlastManager.register(new JQBlast());
        }
    //implement Blast methods
    }
```

The `Blast` engine provides a mechanism to specify the parameters for a search (such as database type, BLAST algorithm type, E-value, etc.) and to submit a sequence into a queue for the actual `Blast` operation. The above design provides a way of accessing an instance of `Blast`, without the need to know the mechanism by which the `Blast` operation is submitted or performed. In this case, `JQBlast` is an implementation of the abstract `Blast` class and that is the one that is instantiated by the `BlastManager`.

When a Java class is loaded, the *Java classloader* will run all the *static* statements first, so a `JQBlast` instance will be created and registered to the `BlastManager`. Now to allow the classloader to load that class we need to call the *java classpath* using the `forName` method from the class `Class`, as shown below:

```
static {
        try {
            Class.forName("org.jfb.jqblast.JQBlast");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
```

Alternately, we can pass the Java class name to the JVM system property using the Java *–D* option and the property name "`blast.driver`", if we don't want to hard code the name of the `Blast` class we would like to use in the code.

```
Java -Dblast.driver=org.jfb.SwingBlast.qblast.QBlast (…)
```

The property is then retrieved using the `getProperty` method as shown below:

```
System.getProperty("blast.driver");
```

We pass the BLAST parameters to the `submitQuery()` method as follows:

```
public Object submitQuery(Map parameters) throws
BlastException {
        String urlapiQuery = createUrlapiQuery(parameters);
        setChanged();
        notifyObservers("Submitting the job to the server
with query\n"
  + urlapiQuery);
        String queryResult = sendQuery(urlapiQuery);

        if (queryResult == null) return null;
        return parseOutReqId(queryResult);
    }
```

The method `createUrlapiQuery()` within `submitQuery()` generates the HTTP-encoded request containing the specified parameters (including the sequence specified by the user (in this case, a test sequence "`AAGTCGATAGCTCGCGCGCCGGCCGTGAGGAAAAAAAAA`").

```
  CMD=Put&QUERY_BELIEVE_DEFLINE=yes&QUERY=%3E+Sequence1%7CDNA
%7C38+bp%0AAGTCGATAGCTCGCGCGCCGGCCGTGAGGAAAAAAAAA&DATABASE=nr
&PROGRAM=blastn&EXPECT=0.001
```

The method is described below:

```
      private String createUrlapiQuery(Map parameters) {
          StringBuffer          query          =          new
StringBuffer("CMD=Put&QUERY_BELIEVE_DEFLINE=yes");
          try {

query.append("&QUERY=").append(URLEncoder.encode((String)
parameters.get("sequenceText"), "UTF-8"))
                        .append("&DATABASE=").append((String)
parameters.get("database"))
                        .append("&PROGRAM=").append((String)
parameters.get("blastType"))
                        .append("&EXPECT=").append((String)
parameters.get("eValue"));
          } catch (UnsupportedEncodingException uee) {
              uee.printStackTrace();
          }
          return query.toString();
      }
```

In this case, the method returns a String object containing the sequence to be submitted for the BLAST search, the database to be searched against, the BLAST program to be used and the cut-off E-value for the search.

The `setChanged()` method in `submitQuery()` is derived from the `Observable` class and is used to keep track of changes in the status of an object, in this case, `Blast`. The `Observable` class notifies changes in states of objects by calling the `notifyObservers()` method. In this example, we will inform the user that a search job has been submitted (with the message, "Submitting the job to the server with query", and appends the `urlapiQuery` string to it.

```
      notifyObservers("Submitting the job to the server with
query\n" + urlapiQuery);
```

Next we send the query for BLAST using the `sendQuery()` method:

```
    private   String   sendQuery(String   httpQuery)   throws
BlastException {
      DataOutputStream printer = null;
      URLConnection urlConnection;
      ByteArrayOutputStream outputStream = null;
```

```
      try {
        urlConnection = new URL(blastUrl).openConnection();
        urlConnection.setDoInput(true);
        urlConnection.setDoOutput(true);
        urlConnection.setUseCaches(false);
        urlConnection.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
        urlConnection.setRequestProperty("Content-Length", ""
+ httpQuery.length());
        printer                    =                    new
DataOutputStream(urlConnection.getOutputStream());
        printer.writeBytes(httpQuery);

        // Read the result
        BufferedReader reader = null;
        reader = new BufferedReader(new
          InputStreamReader(urlConnection.getInputStream()));
        outputStream = new ByteArrayOutputStream();
        String str;
        while ((str = reader.readLine()) != null) {
          outputStream.write(str.getBytes());
        }
      } catch (MalformedURLException mue) {
        mue.printStackTrace();
        throw new BlastException(blastUrl + " is malformed");
      } catch (IOException ioe1) {
        ioe1.printStackTrace();
        throw   new   BlastException("Could   not   get   the
connection or write to it");
      } finally {
        try {
          printer.close();
          printer = null;
        } catch (IOException ignore) {
          ignore.printStackTrace();
        }
      }
      return outputStream != null ? outputStream.toString() :
null;
    }
```

The `sendQuery()` method returns a String carrying the results of the operation:

```
String queryResult = sendQuery(urlapiQuery);
```

We then parse the result (unless no hits were found) using the `parseOutReqId()` method:

```
      if (queryResult == null) return null;
```

```
      return parseOutReqId(queryResult);
```

The `parseOutReqId()` method parses the RID and RTOE from the returned string which is of type:

```
  QBlastInfoBegin                RID    =    1097884888-2134-
17842894979.BLASTQ4   RTOE = 30QBlastInfoEnd
```

and returns the `RequestIdentifier`:

```
  private RequestIdentifier parseOutReqId(String string) {
      String rid = null;
      String rtoe = null;

      try {
        RE              regex              =              new
RE("QBlastInfoBegin(\\s*)RID(\\s*)=\\2(\\S*)(\\s*)RTOE\\2=\\2
(.*)QBlastInfoEnd");
        boolean matched = regex.match(string);

        if (matched) {
          rid = regex.getParen(3);
          rtoe = regex.getParen(5);
        }
      } catch (RESyntaxException ree) {
      }
      if (rid == null || rtoe == null)
        return null;
      return             new            RequestIdentifier(rid,
Integer.parseInt(rtoe));
    }
```

Once we obtain the RID and RTOE, we wait for a period of time specified by the RTOE before trying to access the results.

```
  public Object requestResult(RequestIdentifier identifier)
throws BlastException {
      if (identifier == null)
        throw new BlastException("Cannot get the request
identifier");

      setChanged();
      notifyObservers("Getting from JQBlast Service the RID
("   +   identifier.getRid()   +   ")   and   RTOE   ("   +
identifier.getRtoe() + ").");

      // Wait the rtoe time before sending any request back
to the server
      try {
        long     timeOut     =     identifier.getRtoe()     +
```

```
identifier.getTime();

        if (timeOut > System.currentTimeMillis()) {
          int     timeLeft     =     ((int)     (timeOut     -
System.currentTimeMillis())) * 1000;

          synchronized (this) {
            while (timeLeft > 0) {
              wait(waitTime);
              setChanged();
              notifyObservers("Time left " + ((timeLeft -=
waitTime) / 1000) + "s before requesting the result");
            }
          }
        }

      } catch (InterruptedException ie) {
        ie.printStackTrace();
      }

      setChanged();
      notifyObservers("Requesting the result for rid: " +
identifier.getRid());
      StringBuffer           query           =           new
StringBuffer("CMD=Get&FORMAT_TYPE=XML");
      query.append("&RID=" + identifier.getRid());
      String ri = query.toString();

      String queryResult = null;
      String status = null;

      boolean hasResult = false;
      int ct = 0;
      RE regex = null;
      try {
        regex                          =                    new
RE("QBlastInfoBegin(\\s*)Status=(.*)QBlastInfoEnd");
      } catch (RESyntaxException ree) {

      }

      synchronized (this) {
        while (!hasResult) {
          status = null;
          queryResult = sendQuery(ri);
          boolean matched = regex.match(queryResult);

          if (matched) {
            status = regex.getParen(2);
          }
          hasResult = !"WAITING".equals(status);
          if (hasResult) {
            break;
```

```
          }

          setChanged();
          notifyObservers("Waiting " + NUMBER_OF_SECOND + "
seconds before re-trying (total waiting time: " + (ct +=
NUMBER_OF_SECOND) + "s).");
          try {
            wait(NUMBER_OF_SECOND * 1000);
          } catch (InterruptedException ie1) {
            ie1.printStackTrace();
          }
        }
      }
      if ("UNKNOWN".equals(status)) {
        throw new BlastException("Result for RID " +
identifier.getRid() + " failed.");
      }
      setChanged();
      notifyObservers("Getting back the blast result in
XML");
      return queryResult;
    }
```

The complete code for `JQBlast.java` is shown in **Listing 3.4**.

Listing 3.4. JQBlast.java

```
package org.jfb.jqblast;

import org.apache.regexp.RE;
import org.apache.regexp.RESyntaxException;
import org.jfb.blast.Blast;
import org.jfb.blast.BlastException;
import org.jfb.blast.BlastManager;

import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;
import java.util.HashMap;

public class JQBlast extends Blast {
```

```
    static {
        System.out.println("Registering " + JQBlast.class);
        BlastManager.register(new JQBlast());
    }

    private static final String blastUrl =
"http://www.ncbi.nlm.nih.gov/blast/Blast.cgi";
    private static final int NUMBER_OF_SECOND = 3;

    public Object submitQuery(Map parameters) throws
BlastException {
        String urlapiQuery = createUrlapiQuery(parameters);
        setChanged();
        notifyObservers("Submitting the job to the server
with query\n" + urlapiQuery);
        String queryResult = sendQuery(urlapiQuery);

        if (queryResult == null) return null;
        return parseOutReqId(queryResult);
    }

    final static int waitTime = 2000;

    public Object requestResult(Object identifier) throws
BlastException {
        if (identifier == null || !(identifier instanceof
RequestIdentifier))
            throw new BlastException("Cannot get the
request identifier " + identifier);

        RequestIdentifier rIdentifier = (RequestIdentifier)
identifier;
        setChanged();
        notifyObservers("Getting from JQBlast Service the
RID (" + rIdentifier.getRid()
                + ") and RTOE (" + rIdentifier.getRtoe() +
").");

        // Wait the rtoe time before sending any request
back to the server
        try {
            long timeOut = rIdentifier.getRtoe() +
rIdentifier.getTime();

            if (timeOut > System.currentTimeMillis()) {
                int timeLeft = ((int) (timeOut -
System.currentTimeMillis())) * 1000;

                synchronized (this) {
                    while (timeLeft > 0) {
                        wait(waitTime);
                        setChanged();
                        notifyObservers("Time left " +
```

```
((timeLeft -= waitTime) / 1000) + "s before requesting the
result");
                    }
                }
            }

        } catch (InterruptedException ie) {
            ie.printStackTrace();
        }

        // do a loop every 3 seconds send the request until
we get the status = READY and the blast result
        // End of loop
        setChanged();
        notifyObservers("Requesting the result for rid: " +
rIdentifier.getRid());
        StringBuffer          query          =          new
StringBuffer("CMD=Get&FORMAT_TYPE=XML");
        query.append("&RID=" + rIdentifier.getRid());
        String ri = query.toString();

        String queryResult = null;
        String status = null;

        boolean hasResult = false;
        int ct = 0;
        RE regex = null;
        try {
            regex                    =                    new
RE("QBlastInfoBegin(\\s*)Status=(.*)QBlastInfoEnd");
        } catch (RESyntaxException ree) {
            // We ignore it since we've checked the regex
already!
        }
        Runtime runtime = Runtime.getRuntime();

        synchronized (this) {
            while (!hasResult) {
                status = null;
                queryResult = sendQuery(ri);
                boolean matched = regex.match(queryResult);

                if (matched) {
                    status = regex.getParen(2);
                }
                hasResult = !"WAITING".equals(status);
                if (hasResult) {
                    break;
                }

                setChanged();
                notifyObservers("Waiting          "          +
NUMBER_OF_SECOND
```

```
                        + " seconds before re-trying (total
waiting time: " + (ct += NUMBER_OF_SECOND) + "s). "
                        + runtime.freeMemory() + " bytes
left");
                try {
                    wait(NUMBER_OF_SECOND * 1000);
                } catch (InterruptedException ie1) {
                    ie1.printStackTrace();
                }
            }
        }
        if ("UNKNOWN".equals(status)) {
            throw new BlastException("Result for RID " +
rIdentifier.getRid() + " failed.");
        }
        setChanged();
        String fileName = createTempFileName();
        try {
            OutputStream outputStream = new
FileOutputStream(fileName);
            outputStream.write(queryResult.getBytes());
        } catch (IOException ioe) {
            throw new BlastException("Saving result for RID
" + rIdentifier.getRid()
                    + " into " + fileName + " failed.",
ioe);
        }
        notifyObservers("Getting back the blast result in
XML " + queryResult.length());
        return fileName;
    }

    private String sendQuery(String httpQuery) throws
BlastException {
        DataOutputStream printer = null;
        URLConnection urlConnection;
        ByteArrayOutputStream outputStream = null;
        String fileName = null;

        try {
            urlConnection = new
URL(blastUrl).openConnection();
            urlConnection.setDoInput(true);
            urlConnection.setDoOutput(true);
            urlConnection.setUseCaches(false);
            urlConnection.setRequestProperty("Content-
Type", "application/x-www-form-urlencoded");
            urlConnection.setRequestProperty("Content-
Length", "" + httpQuery.length());
            printer = new
DataOutputStream(urlConnection.getOutputStream());
            printer.writeBytes(httpQuery);
```

```
                // Let's read the result
                BufferedReader reader = null;
                reader      =      new      BufferedReader(new
InputStreamReader(urlConnection.getInputStream()));
                outputStream = new ByteArrayOutputStream();
                String str;
                while ((str = reader.readLine()) != null) {
                    outputStream.write(str.getBytes());
                }
            } catch (MalformedURLException mue) {
                mue.printStackTrace();
                throw  new  BlastException(blastUrl  +  "  is
malformed");
            } catch (IOException ioe1) {
                ioe1.printStackTrace();
                throw new BlastException("Could not get the
connection or write to it");
            } finally {
                try {
                    printer.close();
                    printer = null;
                } catch (IOException ignore) {
                    ignore.printStackTrace();
                }
            }
            return   outputStream   ==   null   ?   null   :
outputStream.toString();
        }

        private RequestIdentifier parseOutReqId(String string)
{
            String rid = null;
            String rtoe = null;

            try {
                // <!--QBlastInfoBegin      RID = 1097884888-
2134-17842894979.BLASTQ4    RTOE = 30QBlastInfoEnd-->
                RE            regex            =            new
RE("QBlastInfoBegin(\\s*)RID(\\s*)=\\2(\\S*)(\\s*)RTOE\\2=\\2
(.*)QBlastInfoEnd");
                boolean matched = regex.match(string);

                if (matched) {
                    rid = regex.getParen(3);
                    rtoe = regex.getParen(5);
                }
            } catch (RESyntaxException ree) {
                // We ignore it since we checked the regex
already!
            }
            if (rid == null || rtoe == null)
                return null;
            return          new          RequestIdentifier(rid,
```

```
Integer.parseInt(rtoe));
      }

      private String createUrlapiQuery(Map parameters) {
          StringBuffer          query          =          new
StringBuffer("CMD=Put&QUERY_BELIEVE_DEFLINE=yes");
          try {

query.append("&QUERY=").append(URLEncoder.encode((String)
parameters.get("sequenceText"), "UTF-8"))
                      .append("&DATABASE=").append((String)
parameters.get("database"))
                      .append("&PROGRAM=").append((String)
parameters.get("blastType"))
                      .append("&EXPECT=").append((String)
parameters.get("eValue"));
          } catch (UnsupportedEncodingException uee) {
              uee.printStackTrace();
          }
          return query.toString();
      }

      private String createTempFileName() {
          return     System.getProperty("java.io.tmpdir")     +
File.separator
                  + "blast-" + System.currentTimeMillis() +
".xml";

      }
      private static String packBy(int i, String s) throws
RESyntaxException {
          String substIn = "[a-zA-Z]{" + i + "}";
          String substTo = "$0 ";
          RE re = new RE(substIn);
          return              re.subst(s,              substTo,
RE.REPLACE_BACKREFERENCES);
      }
  }
```

Enhancing the SwingBlast Application

Let's also take a look at the code that generates the GUI for the application. The `SwingBlast` Version 1.3 we created in the last Chapter is shown in **Fig. 3.5**.

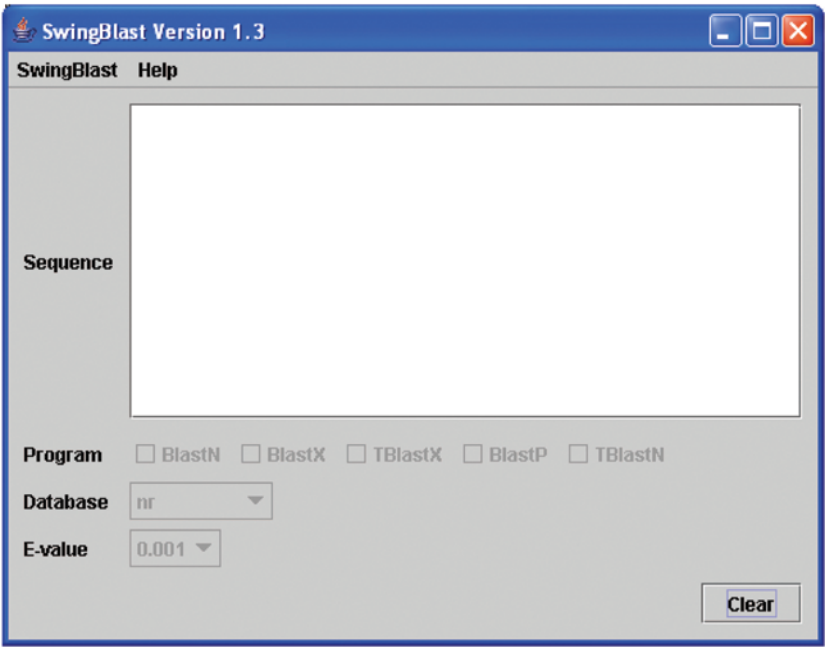

Fig. 3.5. SwingBlast version 1.3

We will enhance `SwingBlast` in a number of ways in this Chapter. In particular, we will:

1. Introduce a Format button to convert the entered sequence into Fasta format. In the earlier version, the `SwingBlast` application required the user to lose focus away from the text area in order to perform the formatting.

2. Add a Submit button to send sequences for BLAST searches.

3. Add code behind the BLAST programs (BLASTN, BLASTX, etc.) so that checking the boxes will enable the user to run the corresponding BLAST programs.

4. Add functionality to prompt the user to save BLAST search results.

We will call the resulting application `SwingBlast` Version 2.1. We add the button widgets we need for the `SwingBlast` application as we did previously.

```
private JButton formatBtn;
formatBtn = new JButton("Format Sequence");
```

To place the button in the GUI, we use the `JPanel` object:

```
JPanel panel = new JPanel();
panel.add(formatBtn);
seqPanel.add(panel, BorderLayout.CENTER);
```

To format a sequence, we first need to know when the text area is populated with a sequence. To do this we implement an event listener, which was explained in Chapter 2.

```
    private void addListeners() {
      formatBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          // Check if sequence is DNA, RNA or protein
          // Retrieve text entered in the text area
          String sequenceText = sequenceArea.getText();
          if (sequenceText == null || sequenceText.length()
== 0) {
            cleanAllParameters();
            return;
          }
        }
      }
    }
```

The `cleanAllParameters()` method clears the text in the text area and disables the `enableFunctions()` method which checks the entered sequence for type, that is, DNA, RNA or protein.

```
    private void cleanAllParameters() {
      sequenceArea.setText("");
      enableFunctions(-1);
    }
```

Next, let's add the code to format the input sequence. We will program the format button to cause the sequence in the text area to be wrapped into lines of 50 bases each and add a Fasta header at the top using the code below:

```
    private StringBuffer format(String sequence) {
        int i = 1;
        final int seqLen = sequence.length();
        StringBuffer sb = new StringBuffer(seqLen);
        if (seqLen > 50) {
            char[] chars = sequence.toCharArray();
```

```
                for (int j = 0; j < chars.length; j++) {
                    sb.append(chars[j]);

                    if (i++ % 50 == 0) {
                        sb.append("\n");
                    }
                }
            } else {
                sb.append(sequence);
            }
            return sb;
    }
```

We had described the logic to program the check boxes for the various BLAST algorithms based on the input sequence earlier in Chapter 2. The application at this stage appears as is shown in **Fig. 3.6**. Let's test the application with a fragment of the human cystic fibrosis transmembrane conductance regulator (CFTR) mRNA sequence (gi: 90421312) we had described in Chapter 2. Compile and run the application and paste the sequence in the text area (**Fig. 3.7**).

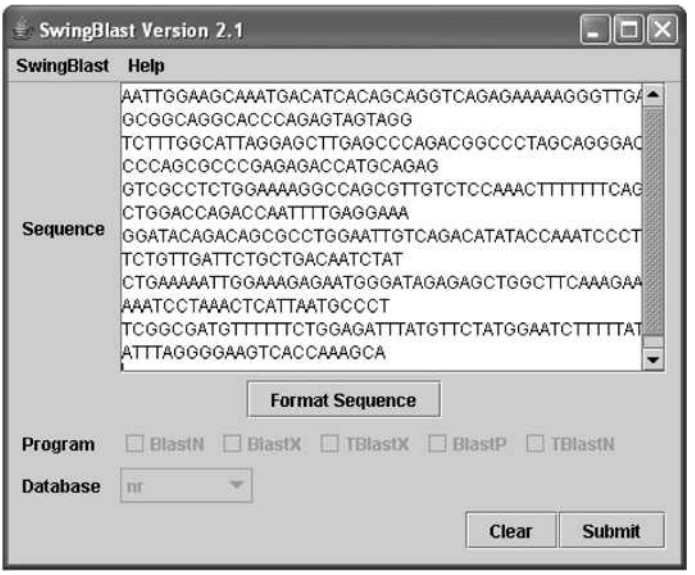

Fig. 3.6. SwingBlast Version 2.1

The formatted sequence is shown below (**Fig. 3.7**).

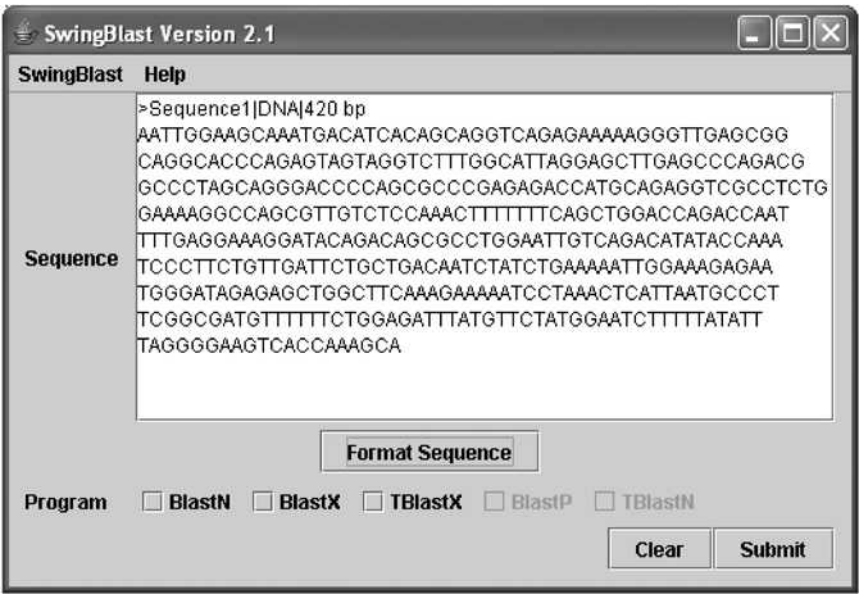

Fig. 3.7. Fasta formatted DNA sequence

To align the Fasta format sequence properly, we had described the use of a monospaced font earlier for the DNA alphabet:

```
            final Font sf = sequenceArea.getFont();
            Font  f  =  new  Font("Monospaced",  sf.getStyle(),
sf.getSize());
            sequenceArea.setFont(f);
```

An explicit monospace font such as *Courier* can also be used provided it is installed on your machine. The application with the sequence formatted in monospace font is shown in **Fig. 3.8**.

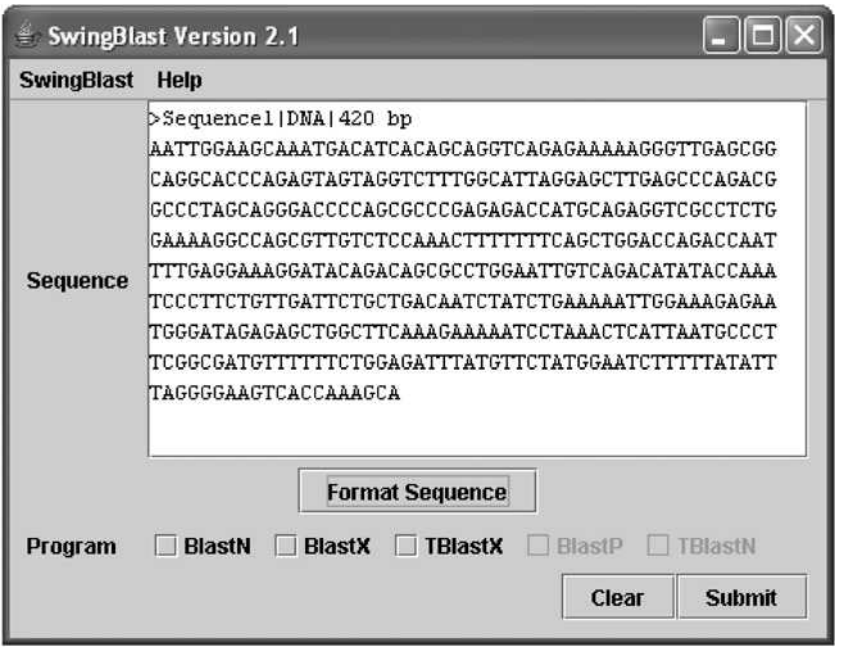

Fig. 3.8. Fasta formatting with a monospace font

Note that the application first checks if the sequence is in Fasta format before applying the formatting. If a sequence that is pasted is already in Fasta format, clicking the "`Format Sequence`" button does not have any effect. The user can now select one or more of the available BLAST options and hit `Submit` to run the search. Let's run a search with the partial CFTR sequence using BLASTN and BLASTX using SwingBlast 2.1 (**Fig. 3.9**).

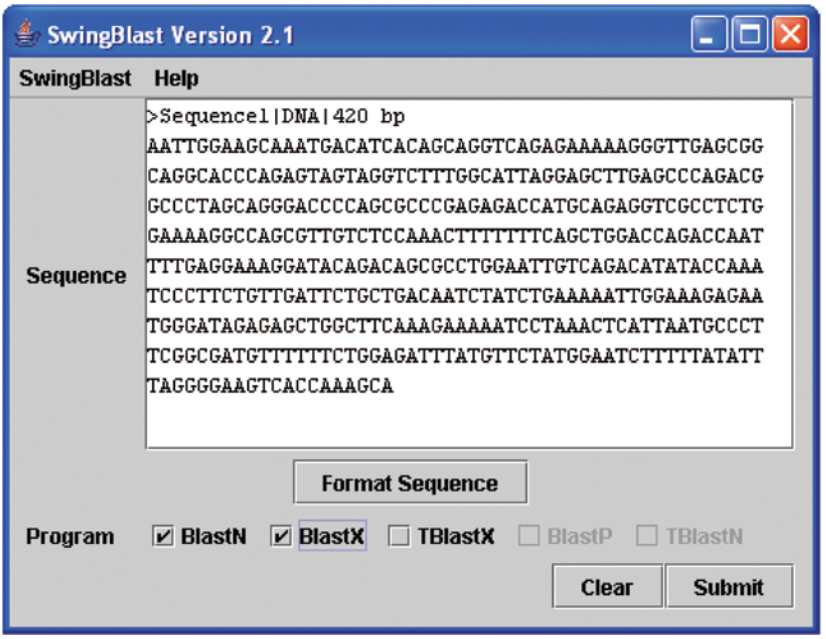

Fig. 3.9. Running a BLASTN and BLASTX search

We get a notification once each of the requested BLAST search is complete as shown below for the BLASTN search (**Fig 3.10**). After each analysis is complete, the application also prompts the user to save the results of the search in a local text file (**Fig. 3.11 - 3.12**).

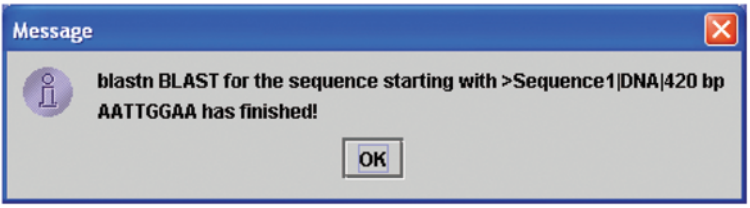

Fig. 3.10. BLAST search status notification

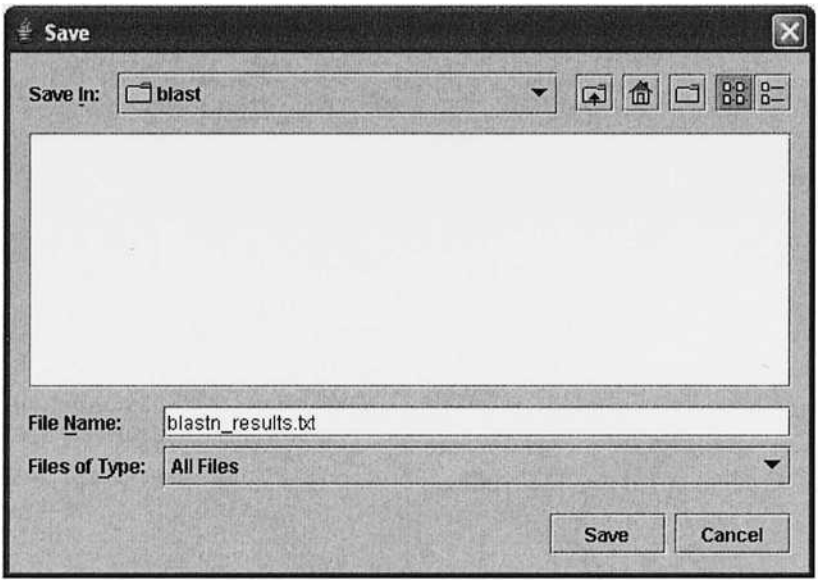

Fig. 3.11. Saving BLAST results in a local file

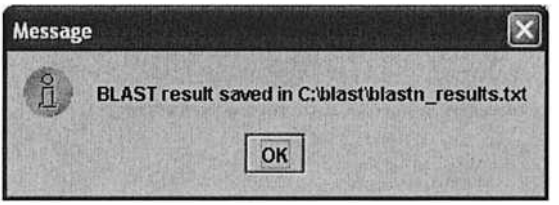

Fig. 3.12. Saving BLAST results in a local file

Note that if a file of that name already exists, the application warns the user and provides an option to overwrite the existing file or save it with a different name (**Fig. 3.13**).

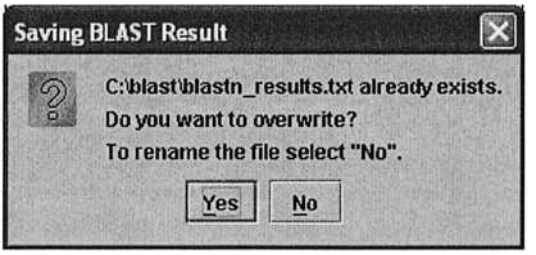

Fig. 3.13. Saving BLAST results in a different file

This functionality is implemented within the `saveBlast()` function as shown in **Listing 3.5**.

Listing 3.5. The saveBlast() function

```
    private void saveBlast(String tmpFileName) {
      final String fileNameFromUser = getFileNameFromUser();
      if (fileNameFromUser == null)
        return;

      final File tmpFile = new File(tmpFileName);
      final File userFile = new File(fileNameFromUser);
      if (userFile.exists()) {
        String errMes = fileNameFromUser + " already
exists.\nDo you want to overwrite?\n" + "To rename the file
select \"No\".";
        int choice = JOptionPane.showConfirmDialog(this,
errMes, "Saving BLAST Result", JOptionPane.YES_NO_OPTION);
        if (choice == JOptionPane.YES_OPTION) {
          userFile.delete();
          tmpFile.renameTo(userFile);
        } else {
          saveBlast(tmpFileName);
        }
      } else {
        tmpFile.delete();
        JOptionPane.showMessageDialog(SequenceForm2_2.this,
"BLAST result saved in " + fileNameFromUser);
      }
    }
```

If the user doesn't want to overwrite an existing file, a new file name must be supplied. This is implemented in the `getFileNameFromUser()` function described below (**Listing 3.6**).

Listing 3.6. The getFileNameFromUser() function

```
    private String getFileNameFromUser() {
      JFileChooser fc = new JFileChooser();
      if (fc.showSaveDialog(this) ==
JFileChooser.APPROVE_OPTION) {
        return fc.getSelectedFile().getAbsolutePath();
      } else {
        return null;
      }
    }
```

The BLAST results can be viewed in their raw format (as saved in the text file above) using a text editor (**Fig. 3.14**) for parsing to diplay the results in a graphical format. The complete code for `SwingBlast` version 2.1 is shown in **Listing 3.7**.

Fig. 3.14. Viewing saved BLAST results in text format

Listing 3.7. SwingBlast Version 2.1

```
package org.jfb.swingblast2;

import org.apache.regexp.RE;
import org.apache.regexp.RESyntaxException;
import org.jfb.blast.Blast;
import org.jfb.blast.BlastException;
import org.jfb.blast.BlastManager;
import org.jfb.jqblast.RequestIdentifier;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Observable;
import java.util.Observer;

public class SwingBlast2_1 extends JFrame {
```

```
    private static final String APP_NAME = "SwingBlast";
    private static final String APP_VERSION = "Version
2.1";

    private static final Dimension LABEL_PREFERRED_SIZE =
new Dimension(57, 16);
    private static final Dimension COMBO_PREFERRED_SIZE =
new Dimension(60, 25);
    private static final Dimension CP_PREF_SIZE = new
Dimension(480, 380);

    private static final int TYPE_DNA = 0;
    private static final int TYPE_RNA = 1;
    private static final int TYPE_PROTEIN = 2;

    private static final String[] BLAST_PROGRAMS_DNA = new
String[]{"BlastN", "BlastX", "TBlastX"};
    private static final String[] BLAST_PROGRAMS_PROTEIN =
new String[]{"BlastP", "TBlastN"};
    private static final String[] DATABASES = new
String[]{"nr", "est_human"};
    private static final String[] EVALUES = new
String[]{"0.001", "0.01", "0.1", "1", "10", "100"};

    private JComponent newContentPane;
    private JTextArea sequenceArea;
    private JScrollPane scrollPaneArea;

    private JCheckBox[] chbDna;
    private JCheckBox[] chbProtein;
    private JComboBox cobDbs;
    private JComboBox cobEvalues;

    private JButton submitBtn;
    private JButton formatBtn;
    private JButton clearBtn;

    private JMenuItem aboutItem;
    private JMenuItem quitItem;
    private static final double SEQ_THRESHOLD = 0.85;
    private static final int TYPE_UNKNOWN = -1;
    private int typeOfSequence;
    private static final String SEQ_HEADER_GEN =
">Sequence1|";
    private static final int SUB_MAX = 30;

    static {
        try {
            Class.forName("org.jfb.jqblast.JQBlast");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
```

```
    public SwingBlast2_1() {
        super();
    }

    private void seqFormInit() {
        setTitle(APP_NAME + " " + APP_VERSION);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        newContentPane = new JPanel();
        newContentPane.setOpaque(true);
        newContentPane.setLayout(new BorderLayout());

        setContentPane(newContentPane);

        // Add the menu bar.
        JMenuBar menu = new JMenuBar();
        JMenu swingBlastMenu = new JMenu(APP_NAME);
        quitItem = new JMenuItem("Quit");
        swingBlastMenu.add(quitItem);
        menu.add(swingBlastMenu);

        JMenu helpMenu = new JMenu("Help");
        aboutItem = new JMenuItem("About");
        helpMenu.add(aboutItem);
        menu.add(helpMenu);
        setJMenuBar(menu);

        // Create the seqLbl pane
        JPanel sequencePanel = new JPanel();
        JLabel seqLbl = new JLabel("Sequence");
        sequenceArea = new JTextArea();
        sequenceArea.setLineWrap(true);
        final Font sf = sequenceArea.getFont();
        Font f = new Font("Monospaced", sf.getStyle(),
sf.getSize());
        sequenceArea.setFont(f);
        scrollPaneArea = new JScrollPane(sequenceArea);
        scrollPaneArea.setPreferredSize(new Dimension(300,
200));
        formatBtn = new JButton("Format Sequence");

        sequencePanel.setLayout(new
BoxLayout(sequencePanel, BoxLayout.LINE_AXIS));
        sequencePanel.add(seqLbl);
        sequencePanel.add(Box.createRigidArea(new
Dimension(10, 0)));
        sequencePanel.add(scrollPaneArea);

        JPanel seqPanel = new JPanel();
        seqPanel.setLayout(new BorderLayout());
        seqPanel.add(sequencePanel, BorderLayout.NORTH);
        JPanel panel = new JPanel();
        panel.add(formatBtn);
```

```
            seqPanel.add(panel, BorderLayout.CENTER);

            //Lay out the buttons from left to right.
            JPanel buttonPane = new JPanel();
            submitBtn = new JButton("Submit");
            clearBtn = new JButton("Clear");

            buttonPane.setLayout(new       BoxLayout(buttonPane,
BoxLayout.LINE_AXIS));
            buttonPane.add(Box.createHorizontalGlue());
            buttonPane.add(Box.createRigidArea(new
Dimension(10, 0)));
            buttonPane.add(clearBtn);
            buttonPane.add(submitBtn);

            JPanel jPanel = new JPanel();
            jPanel.setLayout(new BorderLayout());
            jPanel.setBorder(BorderFactory.createEmptyBorder(0,
10, 10, 10));
            jPanel.add(seqPanel, BorderLayout.NORTH);
            jPanel.add(createProgramPanel(),
BorderLayout.CENTER);
            jPanel.add(buttonPane, BorderLayout.SOUTH);

            newContentPane.add(jPanel, BorderLayout.CENTER);
            newContentPane.setPreferredSize(CP_PREF_SIZE);

            //Display the window.
            pack();
            Dimension                  screenSize                =
Toolkit.getDefaultToolkit().getScreenSize();
            setLocation((screenSize.width - CP_PREF_SIZE.width)
/ 2,
                    (screenSize.height - CP_PREF_SIZE.height) /
2);
            setVisible(true);
            addListeners();
        }

        private JPanel createProgramPanel() {
            // Let's get the program panel using the same
layout
            JPanel programPanel = new JPanel();
            JLabel program = new JLabel("Program");
            program.setPreferredSize(LABEL_PREFERRED_SIZE);
            chbDna = new JCheckBox[BLAST_PROGRAMS_DNA.length];
            String blastProgram;
            for (int i = 0; i < BLAST_PROGRAMS_DNA.length; i++)
{
                blastProgram = BLAST_PROGRAMS_DNA[i];
                chbDna[i] = new JCheckBox(blastProgram);
                chbDna[i].setMaximumSize(COMBO_PREFERRED_SIZE);
            }
```

```
            chbProtein                    =                    new
JCheckBox[BLAST_PROGRAMS_PROTEIN.length];
            for (int i = 0; i < BLAST_PROGRAMS_PROTEIN.length;
i++) {
                blastProgram = BLAST_PROGRAMS_PROTEIN[i];
                chbProtein[i] = new JCheckBox(blastProgram);

chbProtein[i].setMaximumSize(COMBO_PREFERRED_SIZE);
            }

            programPanel.setLayout(new  BoxLayout(programPanel,
BoxLayout.LINE_AXIS));
            programPanel.add(program);
            programPanel.add(Box.createRigidArea(new
Dimension(10, 0)));
            for (int i = 0; i < chbDna.length; i++) {
                programPanel.add(chbDna[i]);
                programPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
            }
            for (int i = 0; i < chbProtein.length; i++) {
                programPanel.add(chbProtein[i]);
                if (i + 1 < chbProtein.length)
                    programPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
            }
            programPanel.add(Box.createHorizontalGlue());
            JPanel paramPanel = new JPanel();
            paramPanel.setLayout(new      BoxLayout(paramPanel,
BoxLayout.PAGE_AXIS));

            paramPanel.add(programPanel);
            paramPanel.add(Box.createRigidArea(new Dimension(0,
5)));

            // Create the database panel using the same layout
            JPanel databasePanel = new JPanel();
            JLabel database = new JLabel("Database");
            database.setPreferredSize(LABEL_PREFERRED_SIZE);
            cobDbs = new JComboBox(DATABASES);
            cobDbs.setMaximumSize(COMBO_PREFERRED_SIZE);

            databasePanel.setLayout(new
BoxLayout(databasePanel, BoxLayout.LINE_AXIS));
            databasePanel.add(database);
            databasePanel.add(Box.createRigidArea(new
Dimension(10, 0)));
            databasePanel.add(cobDbs);
            databasePanel.add(Box.createHorizontalGlue());
            paramPanel.add(databasePanel);
            paramPanel.add(Box.createRigidArea(new Dimension(0,
5)));
```

```
            // Create the E-Value panel using the same layout
            JPanel evaluePanel = new JPanel();
            JLabel eValue = new JLabel("E-value");
            eValue.setPreferredSize(LABEL_PREFERRED_SIZE);
            cobEvalues = new JComboBox(EVALUES);
            cobEvalues.setMaximumSize(COMBO_PREFERRED_SIZE);

            evaluePanel.setLayout(new    BoxLayout(evaluePanel,
BoxLayout.LINE_AXIS));
            evaluePanel.add(eValue);
            evaluePanel.add(Box.createRigidArea(new
Dimension(10, 0)));
            evaluePanel.add(cobEvalues);
            evaluePanel.add(Box.createHorizontalGlue());
            paramPanel.add(evaluePanel);
            paramPanel.add(Box.createRigidArea(new Dimension(0,
5)));


            enableFunctions(TYPE_UNKNOWN);
            return paramPanel;
        }

        private void addListeners() {
            quitItem.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.exit(0);
                }
            });

            aboutItem.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

JOptionPane.showMessageDialog(SwingBlast2_1.this,  APP_NAME  +
" " + APP_VERSION,
                                "About       "       +       APP_NAME,
JOptionPane.INFORMATION_MESSAGE);
                }
            });

            submitBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    StringBuffer       errMes       =       new
StringBuffer("<HTML>Please      provide      the      following
parameters:<BR>");
                    String sequence = sequenceArea.getText();
                    boolean misPar = false;
                    if (sequence == null || sequence.length()
== 0) {
                        errMes.append("- Sequence<BR>");
                        misPar = true;
                    }
```

```
                    String       database       =       (String)
cobDbs.getSelectedItem();
                    String[] blastTypes = getBlastTypes();
                    if (blastTypes == null || blastTypes.length
== 0) {
                        errMes.append("- blast<BR>");
                        misPar = true;
                    }
                    final String endOfPleaseMes = "</html>";
                    errMes.append(endOfPleaseMes);
                    if (misPar) {

JOptionPane.showMessageDialog(SwingBlast2_1.this, errMes);
                        return;
                    }
                    String       evalue       =       (String)
cobEvalues.getSelectedItem();
                    runBlasts(sequence,  blastTypes,  database,
evalue);
                }
            });

            clearBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    cleanAllParameters();
                }
            });

            formatBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    // Check sequence type
                    // Retrieve text entered in the text area
                    final      String      sequenceText      =
sequenceArea.getText();
                    if     (sequenceText     ==     null     ||
sequenceText.length() == 0) {
                        cleanAllParameters();
                        return;
                    }
                    // Format sequence in FASTA format
                    int idx = sequenceText.indexOf(">");
                    final boolean fastaFormatted = idx != -1;
                    String header = null;
                    String sequence = "";

                    if (fastaFormatted) {
                        int             returnIdx             =
sequenceText.indexOf("\n");

                        if (returnIdx != -1) {
                            header  =  sequenceText.substring(0,
returnIdx);
                            sequence                          =
```

```
sequenceText.substring(returnIdx + 1,
sequenceText.length()).replaceAll("\\s", "").toLowerCase();
                    }
                    // Check if sequence entered
                    updateSequenceArea(header, sequence,
fastaFormatted);
                } else {
                    updateSequenceArea(SEQ_HEADER_GEN,
sequenceText.toLowerCase(), fastaFormatted);
                }
            }
        });
    }

    private void updateSequenceArea(String header, String
sequence, boolean fastaFormatted) {
        String seqText;
        if (sequence.length() == 0)
            return;

        // Retrieve sequence type
        this.typeOfSequence = TYPE_UNKNOWN;
        try {
            this.typeOfSequence =
getSequenceType(sequence);
        } catch (RESyntaxException rese) {
            rese.printStackTrace();
        }

        String type = null;
        String unitOfLength = null;

        switch (this.typeOfSequence) {
            case TYPE_DNA:
                type = "DNA";
                unitOfLength = " bp";
                break;
            case TYPE_RNA:
                type = "RNA";
                unitOfLength = " bp";
                break;
            case TYPE_PROTEIN:
                type = "Protein";
                unitOfLength = " aa";
                break;
            default:
                type = "N/A";
                unitOfLength = " N/A";
        }

        if (!fastaFormatted) {
            seqText = header + type + "|" +
sequence.length() + unitOfLength + "\n" +
```

```
format(sequence.toUpperCase());
            } else {
                seqText      =      header      +      "\n"      +
format(sequence.toUpperCase());
            }

            // Display results in the sequence area
            sequenceArea.setText(seqText);

            enableFunctions(this.typeOfSequence);
        }

        private StringBuffer format(String seq) {
            int i = 1;
            String sequence = seq.replaceAll("\n", "");
            final int seqLen = sequence.length();
            StringBuffer sb = new StringBuffer(seqLen);
            if (seqLen > 50) {
                char[] chars = sequence.toCharArray();
                for (int j = 0; j < chars.length; j++) {
                    sb.append(chars[j]);

                    if (i++ % 50 == 0) {
                        sb.append("\n");
                    }
                }
            } else {
                sb.append(sequence);
            }
            return sb;
        }

        private void runBlasts(final String sequence, String[]
blastTypes, String database, String evalue) {
            Map param = new HashMap();
            param.put("sequenceText", sequence);
            param.put("database", database);
            param.put("eValue", evalue);

            final Observer observer = new Observer() {
                public void update(Observable o, Object arg) {
                    System.out.println("" + arg);
                }
            };

            try {
                for (int i = 0; i < blastTypes.length; i++) {
                    final String blastType = blastTypes[i];
                    final Map tmp = new HashMap(param);
                    tmp.put("blastType", blastType);
                    Thread t = new Thread(new Runnable() {
                        public void run() {
                            try {
```

```
                                final     Blast     blast     =
BlastManager.createBlast();
                                blast.addObserver(observer);
                                RequestIdentifier
requestIdentifier             =            (RequestIdentifier)
blast.submitQuery(tmp);
                                final    String    fileName    =
blast.requestResult(requestIdentifier).toString();
                                final StringBuffer sb =
                                        new
StringBuffer().append(blastType).append("   BLAST   for   the
sequence starting with ")

.append(sequence.length() > SUB_MAX ? sequence.substring(0,
SUB_MAX) : sequence).append(" has finished!");
                                Runnable    runnable    =    new
Runnable() {
                                    public void run() {

JOptionPane.showMessageDialog(SwingBlast2_1.this,
sb.toString());
                                        saveBlast(fileName);
                                    }
                                };

SwingUtilities.invokeLater(runnable);
                            } catch (BlastException be) {
                                be.printStackTrace();
                            } catch (Throwable e) {
                                e.printStackTrace();
                            }
                        }
                    });
                    t.start();
                }
            } catch (Throwable e) {
                e.printStackTrace();
            }
        }

        private void saveBlast(String tmpFileName) {
            final        String        fileNameFromUser        =
getFileNameFromUser();
            if (fileNameFromUser == null)
                return;

            final File tmpFile = new File(tmpFileName);
            final File userFile = new File(fileNameFromUser);
            String finalName = tmpFileName;
            if (userFile.exists()) {
                String errMes = fileNameFromUser + " already
exists.\nDo you want to overwrite?.";
                int                    choice                =
```

```
JOptionPane.showConfirmDialog(this, errMes, "Saving BLAST
Result", JOptionPane.YES_NO_OPTION);
                if (choice == JOptionPane.YES_OPTION) {
                    boolean renamed =
tmpFile.renameTo(userFile);
                    if (renamed) {
                        tmpFile.delete();
                        finalName = fileNameFromUser;
                    }
                } else {
                    saveBlast(tmpFileName);
                    return;
                }
            } else {
                boolean renamed = tmpFile.renameTo(userFile);
                if (renamed) {
                    tmpFile.delete();
                    finalName = fileNameFromUser;
                }
            }
            JOptionPane.showMessageDialog(SwingBlast2_1.this,
"BLAST result saved in " + finalName);
        }

        private String getFileNameFromUser() {
            JFileChooser fc = new JFileChooser();
            if (fc.showSaveDialog(this) ==
JFileChooser.APPROVE_OPTION) {
                return fc.getSelectedFile().getAbsolutePath();
            } else {
                return null;
            }
        }

        protected void finalize() throws Throwable {
            super.finalize();

        }

        private void cleanAllParameters() {
            sequenceArea.setText("");
            enableFunctions(-1);
        }

        private String[] getBlastTypes() {
            JCheckBox[] allTypes = typeOfSequence == TYPE_DNA
|| typeOfSequence == TYPE_RNA
                    ? chbDna : typeOfSequence == TYPE_PROTEIN ?
chbProtein : null;
            if (allTypes == null) return null;

            ArrayList types = new ArrayList();
            for (int i = 0; i < allTypes.length; i++) {
```

```
                JCheckBox cb = allTypes[i];
                if (cb.isSelected())
                    types.add(cb.getText().toLowerCase());
            }
            final String[] res = new String[types.size()];
            types.toArray(res);
            return res;
        }

        private void enableFunctions(int typeOfSequence) {
            if (typeOfSequence == TYPE_DNA || typeOfSequence ==
TYPE_RNA) {
                setChb(chbDna, true);
                setChb(chbProtein, false);
                setCob(cobDbs, true);
                setCob(cobEvalues, true);
            } else if (typeOfSequence == TYPE_PROTEIN) {
                setChb(chbProtein, true);
                setChb(chbDna, false);
                setCob(cobDbs, true);
                setCob(cobEvalues, true);
            } else {
                setChb(chbProtein, false);
                setChb(chbDna, false);
                setCob(cobDbs, false);
                setCob(cobEvalues, false);
            }
        }

        private static void setChb(JCheckBox[] boxes, boolean
value) {
            for (int i = 0; i < boxes.length; i++) {
                boxes[i].setEnabled(value);
                boxes[i].setSelected(false);
            }
        }

        private static void setCob(JComboBox component, boolean
value) {
            component.setEnabled(value);
            component.setSelectedIndex(0);
        }

        public static int getSequenceType(String sequence)
throws RESyntaxException {
            RE re = new RE("[actgnACGTN]+");
            String[] strings = re.split(sequence);
            int numbOfLettersOtherThanATGCNs = 0;

            for (int i = 0; i < strings.length; i++) {
                numbOfLettersOtherThanATGCNs                    +=
strings[i].length();
            }
```

```
            int length = sequence.length();
            int       numbOfACGTNs       =       length       -
numbOfLettersOtherThanATGCNs;

            re = new RE("[uU]+");
            strings = re.split(sequence);
            int numbOfLettersOtherThanUs = 0;

            for (int i = 0; i < strings.length; i++) {
                numbOfLettersOtherThanUs                      +=
strings[i].length();
            }
            int     numbOfUs     =     sequence.length()     -
numbOfLettersOtherThanUs;

            if (numbOfACGTNs / (double) length > SEQ_THRESHOLD)
{
                return TYPE_DNA;
            } else if ((numbOfACGTNs + numbOfUs) / (double)
length > SEQ_THRESHOLD) {
                return TYPE_RNA;
            } else {
                return TYPE_PROTEIN;
            }
        }

        public static void main(String[] args) {
            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    final  SwingBlast2_1  sequenceForm  =  new
SwingBlast2_1();
                    sequenceForm.seqFormInit();
                }
            });
        }
    }
```

Retrieving Sequences From GenBank Using BioJava

Frequently, users know GI numbers of sequences that they use regularly in their research and it is normal for them to submit a GI number of the corresponding sequence for BLAST searches on the NCBI BLAST service. We will next implement a feature in `SwingBlast` whereby users can retrieve a sequence from GenBank based on its GI number. We will use existing *BioJava* routines to retrieve sequences corresponding to a GenBank ID that users may enter into the sequence field. We will need the following BioJava libraries to accomplish this task:

```
org.biojava.bio.seq.Sequence;

org.biojava.bio.seq.db.GenbankSequenceDB;

org.biojava.bio.seq.io.SeqIOTools;
```

These libraries can be obtained from the BioJava website (Binary for J2SE 1.4 or later, as of this writing) at the following URL:

```
http://biojava.org/wiki/BioJava:Download
```

Since users have the option of entering sequences directly into the sequence field, we need to first test if the entered text is a sequence or a genbank ID. We will do this using *regular expressions* as outlined below:

```
text = text.replaceAll("\\s", "");
RE re = null;
try {
    re = new RE("[0-9]+");
} catch (RESyntaxException e1) {
    e1.printStackTrace();
}

boolean isGenBankID = re.match(text);
```

We then create a new instance of the class `GenbankSequenceDB` that will retrieve the Genbank record. `SeqObject` contains the entire GenBank record, that is, the header information, any sequence features and annotation and the actual nucleotide or amino acid sequence.

```
seqObject = genbankSequenceDB.getSequence(text);
```

To see the content of the sequence object retrieved we can write it to the system output using `SeqIOTools` as followed:

```
SeqIOTools.writeGenbank(System.out, seqObject);
```

To grab only the sequence we then use the method `seqString()` from the `seqObject`.

```
sequence = seqObject.seqString();
```

The complete code is as follows:

```
import org.biojava.bio.seq.Sequence;
import org.biojava.bio.seq.db.GenbankSequenceDB;
```

```
  import org.biojava.bio.seq.io.SeqIOTools;

  text = text.replaceAll("\\s", "");
  RE re = null;
  try {
      re = new RE("[0-9]+");
  } catch (RESyntaxException e1) {
      e1.printStackTrace();
  }

  boolean isGenBankID = re.match(text);

  if (isGenBankID) {
      GenbankSequenceDB      genbankSequenceDB      =      new
GenbankSequenceDB();
      header = "GI:" + text;
      Sequence seqObject = null;
      try {
          seqObject = genbankSequenceDB.getSequence(text);
          SeqIOTools.writeGenbank(System.out, seqObject);
      } catch (Exception e) {
          e.printStackTrace();
      }
      sequence = seqObject.seqString();
  }
```

The "Format Sequence" in the application will now have a dual function when a GI number is pasted in the text area – it will retrieve the sequence from GenBank and simultaneously convert it into the Fasta format. We will call this version of the application `SwingBlast` version 2.2. The code for SwingBlast Version 2.2 with this feature implemented is shown in **Listing 3.8**.

Listing 3.8. SwingBlast Version 2.2

```
  Runnable runnable = new Runnable() {
      public void run() {
          String seq = null;
          final          boolean          isGenBankID          =
GenbankDB.isGenBankId(sequenceText);

          if (isGenBankID) {
             boolean canGetSeq = true;
             GenbankSequenceDB    genbankSequenceDB    =    new
GenbankSequenceDB();
             header = "GI:" + text;
             Sequence seqObject = null;
             try {
```

```
                seqObject                                          =
genbankSequenceDB.getSequence(text);
                SeqIOTools.writeGenbank(System.out,
seqObject);
            } catch (Exception e) {
                e.printStackTrace();
            }
            seq = seqObject.seqString();
            if (seq == null || seq.length() == 0 ||
!canGetSeq) {

JOptionPane.showMessageDialog(SwingBlast2_2.this,
                    "Cannot get the sequence for GenBank ID
" + sequenceText);
            return;
        }
    }

    SwingBlast2_2.this.sequence = seq;
    Runnable runnableAwt = new Runnable() {
        public void run() {
            String seqFin = SwingBlast2_2.this.sequence;
            String header = null;
            String sequence = "";

            if (isGenBankID) {
                int i = seqFin.indexOf("\n");
                header = seqFin.substring(0, i);
                sequence     =     seqFin.substring(i     +
"\n".length(), seqFin.length());
            } else {
                sequence = sequenceText.toLowerCase();
                header = SEQ_HEADER_GEN;
            }
            // We first check that there is something.
            updateSequenceArea(header,            sequence,
fastaFormatted, isGenBankID);
        }
    };
    SwingUtilities.invokeLater(runnableAwt);
};
new Thread(runnable).start();
```

Fig. 3.15 and **Fig. 3.16** below show the results of pasting a GenBank Id in the sequence area of SwingBlast Version 2.2.

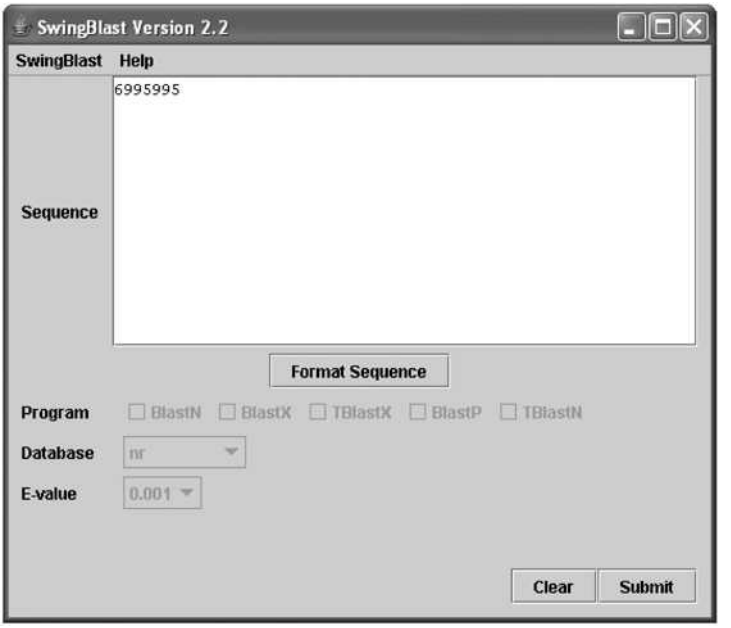

Fig. 3.15. Pasting GI number in the text area for sequence retrieval

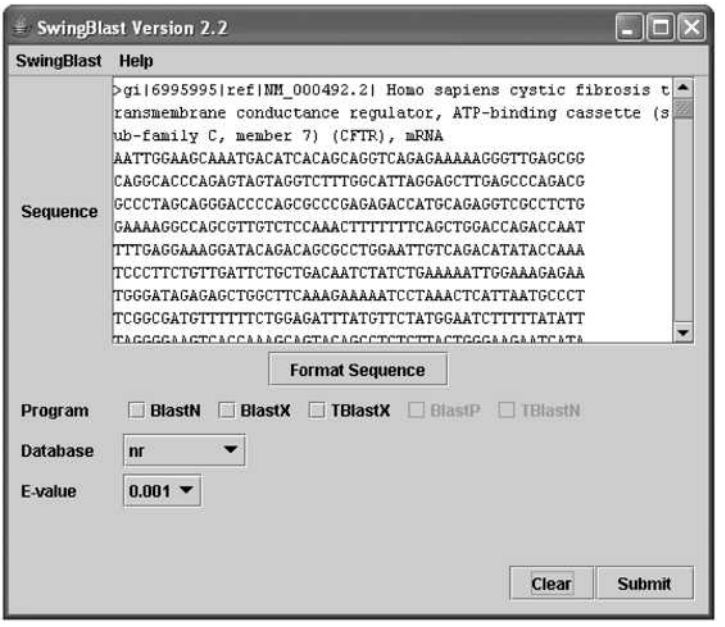

Fig. 3.16. Retrieving a sequence from GenBank from its GI number

Although the Fasta header in **Fig. 3.16** appears to run over multiple lines, it is actually a single line that has wrapped over because of the size of the text area.

Retrieving GenBank Without BioJava

This is how one would implement the retrieval of the sequence using GenBank ID and NCBI web application using regular expressions to parse out the sequence. To implement the retrieval of sequences from GenBank by GI numbers we create a package called `org.jfb.util.GenbankDB`.

The `GenbankDB` class implements a method called `getSequence()` to retrieve sequences from GenBank through requests sent to the following URL (as defined in the String constant GENBANK_URL):

```
"http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?dopt=fasta&
list_uids=";
```

Since the GenBank Id is a number is a numeral, the method performs checks if the user entered GI number is a valid entry. The `getSequence()` method takes a single parameter – the GenBank ID – opens a connection to the URL, obtains the data from GenBank, and performs the necessary parsing, formatting and trimming to get the actual GenBank sequence. To retrieve the CFTR sequence from GenBank using its GI number (6995995, replaced by 90421312), for example, the URL we would use in a browser would be: <<<here

```
http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?dopt=fasta&l
ist_uids=6995995
```

This opens up the GenBank page with the sequence displayed in Fasta format (**Fig. 3.17**). This record needs to be parsed to extract the raw sequence from the HTML formatting on the page. This is easily done since the sequence is bounded by the <pre> and </pre> starting and ending tags. **Fig. 3.18** shows the source HTML of the page with beginning <pre> tag just before the Fasta formatted sequence starts.

Fig. 3.17. GenBank record for the CFTR mRNA sequence

Fig. 3.18. Parsing the raw sequence data from a GenBank record

The code for the GenbankDB class is described in **Listing 3.9.**

Listing 3.9. The GenbankDB class

```
package org.jfb.util;
import org.apache.regexp.RE;
import org.apache.regexp.RESyntaxException;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;

public class GenbankDB {
    private static final String GENBANK_URL =
"http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?dopt=fasta&li
st_uids=";

    public static String getSequence(String gbId) throws
IOException, IllegalArgumentException {
        // A GenBank ID is always a number
        boolean isGenBankID = isGenBankId(gbId);
        String genBankId = gbId.replaceAll("\n", "");

        if (!isGenBankID)
            throw new IllegalArgumentException(genBankId +
" is not a valid GenBank ID");

        BufferedReader reader = null;
        StringBuffer sb;

        try {
            URL url = new URL(GENBANK_URL + genBankId);
            reader = new BufferedReader(new
InputStreamReader(url.openConnection().getInputStream()));
            String s;
            sb = new StringBuffer();
            while ((s = reader.readLine()) != null) {
                sb.append(s + "\n");
            }
        } finally {
            if (reader != null)
                reader.close();
        }

        String tmp = sb.toString().toLowerCase();
        int idx = tmp.indexOf("<pre>");
        int endIdx = tmp.indexOf("</pre>");

        if (idx == -1 || endIdx == -1)
            return null;
```

```
            return    sb.substring(idx    +    "<pre>".length(),
endIdx);
        }

        private static final int CUT_OFF = 30;

        public static boolean isGenBankId(String gbId) {
            RE re = null;
            try {
                re = new RE("([0-9])+");
            } catch (RESyntaxException e1) {

            }

            boolean valid = true;
            String cleanSeq = gbId.replaceAll("\n", "");
            int len = cleanSeq.length();
            final int min = Math.min(CUT_OFF, len);
            String seqPiece = cleanSeq.substring(0, min);

                re.match(seqPiece);
                String match = re.getParen(0);
                valid      =      match      !=      null      &&
match.equals(seqPiece);
            return valid && min == len;
        }
    }
```

Input Validation

Note that there is no input validation in `SwingBlast` 2.2. `SwingBlast` 2.2 does not flag an error when bad characters are present in the sequence entered in the text area. **Fig. 3.19** shows the application behavior when an amino acid ("D") is inserted in what is apparently a nucleotide sequence. The sequence type is deduced as "N/A" because the application cannot determine the sequence type (**Listing 3.7**). For the same reason, none of the BLAST options are available. With the appropriate input validation, the application can catch errors in the entered sequence type and warn the user to make the appropriate changes.

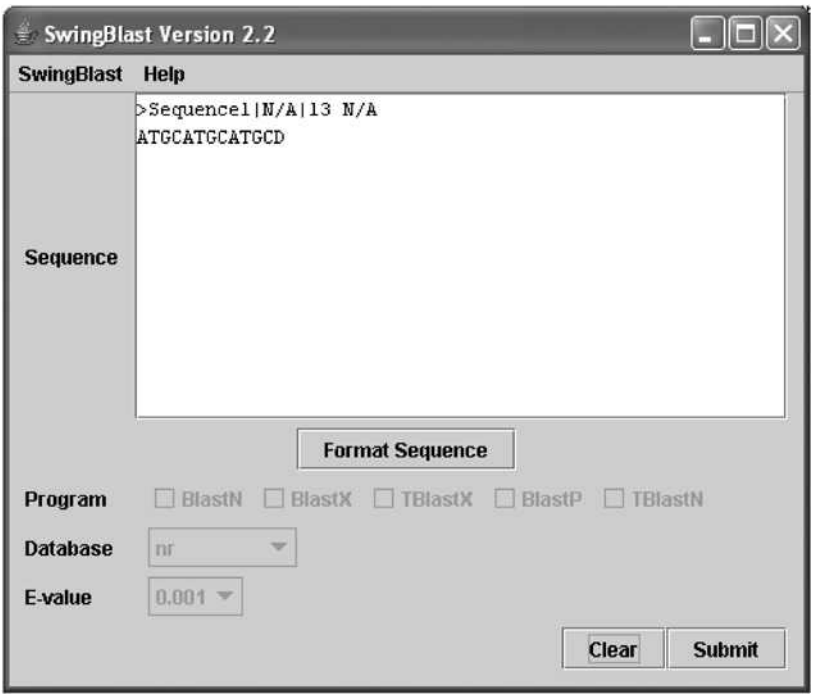

Fig. 3.19. Handling bad characters in input sequence

We will incorporate input validation for a few simple situations as described below:

1. The sequence contains bad characters, that is, characters other than the single letter codes for nucleotides and amino acids. We had illustrated how we used information on the composition of sequences found in nature to determine sequence type in Chapter 2. According to this algorithm, if:

    a. Total number of A, T, G and C's divided by the total length of the sequence is greater that 0.85, it is a DNA sequence

    b. Total number of A, T, G, C and U's divided by the total length of the sequence is greater that 0.85, it is an RNA sequence

If neither of these two conditions are met, the sequence is assumed to be a protein sequence. Again, we are not using the extended DNA/RNA alphabet that includes symbols for sequence ambiguity as defined in the

IUPAC-IUB nucleotide and amino acid nomenclature. Instead, we are illustrating input validation for the simplest of cases where the DNA alphabet is assumed to be composed of A, T, G, C and N and RNA is assumed to be A, U, G, C, N (where N = any nucleotide base) and amino acid alphabet is assumed to be A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W and Y.

Once we have determined the sequence to be DNA, RNA or protein, we check if any bad characters are present in the sequence and warn the user accordingly. We also check if a number instead of a sequence has been entered in the text area. This may very well be a GI number. If it is indeed a GI number, the application will download the corresponding sequence from GenBank when the user presses the "Format Sequence" button. If none of the above conditions are met, the application will print an error message asking the user to check the validity of the sequence or data entered. To add input validation to the SwingBlast application, we add a method called `isValidSequence()`. The method takes the input sequence as the parameter and performs the appropriate checks as described earlier using regular expressions:

```
    private static boolean isValidSequence(String seq) {
      int idx = seq.indexOf(">");
      int idxEndOfFastaHeader = seq.indexOf("\n");
      String sequenceToCheck = null;
      if (idx != -1) {
        sequenceToCheck = seq.substring(idxEndOfFastaHeader +
1, seq.length());
      } else {
        sequenceToCheck = seq;
      }
      return matchRegex(REGEX_DNA, sequenceToCheck)
        || matchRegex(REGEX_RNA, sequenceToCheck)
        || matchRegex(REGEX_PROTEIN, sequenceToCheck)
        || matchRegex(REGEX_GENBANK_ID, sequenceToCheck);
    }
```

The regular expression matching within `matchRegex()` method checks for the following valid patterns:

```
    private static final String REGEX_DNA = "[acgtnACTGN]+";
    private static final String REGEX_RNA = "[acgunACUGN]+";
    private static final String REGEX_PROTEIN =
"[acdefghiklmnpqrstvwyACDEFGHIKLMNPQRSTVWY]+";
    private static final String REGEX_GENBANK_ID = "[0-9]+";
```

The `matchRegex()` method itself is as follows:

```
    private static boolean matchRegex(String regex, String
sequence) {
      RE re = null;
      try {
        re = new RE(regex);
      } catch (RESyntaxException res) {
        // The regex has been tested so no need to chech the
        // exception here
      }

      String cleanSeq = sequence.replaceAll("\n", "");
      boolean valid = true;
      int len = cleanSeq.length();
      int pvsIdx = 0, nextIdx;
      for (int i = 0; i < len; i += CUT_OFF) {
        nextIdx = Math.min(i + CUT_OFF, len);
        String    seqPiece    =    cleanSeq.substring(pvsIdx,
nextIdx);
        re.match(seqPiece);
        String match = re.getParen(0);
        valid = match != null && match.equals(seqPiece);

        if (!valid)
          break;
        pvsIdx = nextIdx;
      }
      return valid;
    }
```

Next we call the `isValidSequence()` method in the `actionPerformed()` event method:

```
        public void actionPerformed(ActionEvent e) {
          // Check sequence type
          // Retrieve text entered in the text area
          final String sequenceText = sequenceArea.getText();
          if (sequenceText == null || sequenceText.length()
== 0) {
            cleanAllParameters();
            return;
          }

          if (!isValidSequence(sequenceText)) {
            JOptionPane.showMessageDialog(SwingBlast2_2.this,
              "The sequence you've entered is neither a DNA
or protein sequence nor a FASTA formatted sequence.\n" +
              "Please provide a valid sequence.");
            return;
          }
```

The application is now able to detect errors in the entered sequence and warn the user with the appropriate message (**Fig. 3.20**).

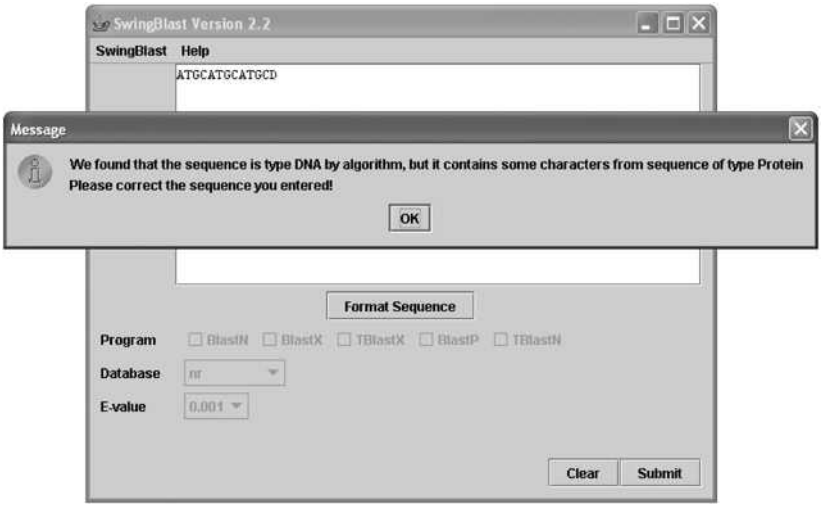

Fig. 3.20. SwingBlast 2.2 with input validation

Fig. 3.21 shows that the application recognizes that just a Fasta header has been provided and results in an error.

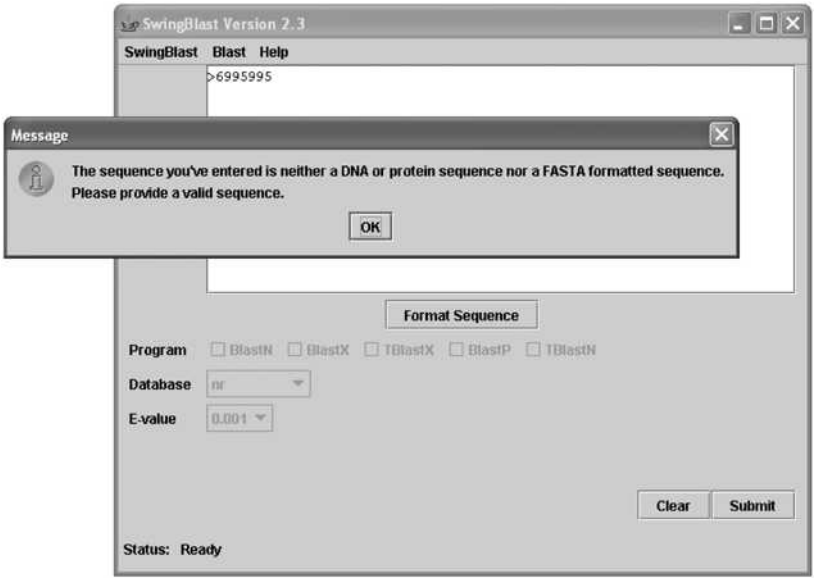

Fig. 3.21. Input validation for wrong GI number format

The application, however, does retrieve the correct sequence information from GenBank if a GI number is provided.

Controlling Program Events and Responses

We will next incorporate some program flow features in `SwingBlast` 2.2. We will program the "`Format Sequence`" button to be enabled only when a non-Fasta formatted sequence is entered in the text area. The format button will be disabled when the application starts and also under the following conditions:

1. When no sequence is available in the text area
2. If the sequence is already in Fasta format (at the time of pasting or right after the sequence is Fasta formatted)
3. When the "`Clear`" button is pressed
4. When the "`Format Sequence`" button is pressed

Let's enhance our application with these features in mind. We will call this `SwingBlast` Version 2.3. The code to enable or disable the "`Format Sequence`" button to meet condition #1 stated above is straight forward as shown in the **Listing 3.10** below. We implement a document listener interface and associate it with the text area widget. Within the document listener, we implement the `insertUpdate()` and `removeUpdate()` methods to respond to events that either insert or modify text within the text area. **Fig. 3.22** shows the `SwingBlast` 2.3 application with the format button disabled at launch.

Listing 3.10. Enabling and disabling the Format button

```
private void addListeners() {
  docListener = new DocumentListener() {
    public void insertUpdate(DocumentEvent e) {
      String text = sequenceArea.getText();
      if (text == null || text.length() == 0) {
        enableFunctions(-1);
        formatBtn.setEnabled(false);
      } else
        formatBtn.setEnabled(true);
    }
```

```
        public void removeUpdate(DocumentEvent e) {
          String text = sequenceArea.getText();
          if (text == null || text.length() == 0) {
            enableFunctions(-1);
            formatBtn.setEnabled(false);
          }
        }

        public void changedUpdate(DocumentEvent e) {
        }
      };

sequenceArea.getDocument().addDocumentListener(docListener);
```

Similarly, to meet condition 2, we include the following code:

```
  final boolean fastaFormatted = sequenceText.indexOf(">") !=
-1;
  formatBtn.setEnabled(!fastaFormatted);
```

For condition 3, the code is as follows:

```
  clearBtn.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
      cleanAllParameters();
    }
  });
```

The cleanAllParameters() method will empty the sequenceArea and by doing this the document listener shown ealier will disable the formatButton as shown below:

```
  private void cleanAllParameters() {
       sequenceArea.setText("");
       enableFunctions(-1);
  }
```

Finally, to meet condition 4, when the button is pressed, the action listener is actually disabling the format button:

```
  formatBtn.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent e) {
              ...

              formatBtn.setEnabled(false);
       }
```

```
});
```

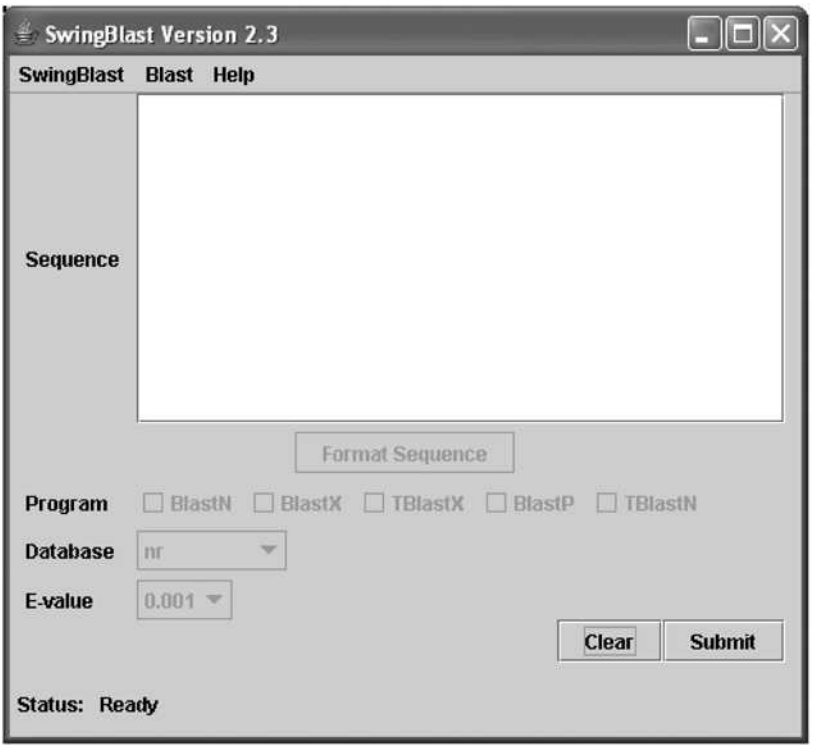

Fig. 3.22. Format button is disabled at start-up

Reporting BLAST Status

SwingBlast allows users to send sequences for multiple simultaneously BLAST analyses. It would be very informative to the user if the application were to provide a status of the current job that it is performing. In the next version of the application, we will add a program status bar to do so. With the program status code in place, the application will provide the user a running status of the jobs in process. Note that these messages will be relayed directly from the QBlast service and printed on the status bar using the observable method discussed earlier. The `SwingBlast` 2.3 application starts with the "`Status: Ready`" message at the bottom left of the application window as shown in **Fig. 3.23**. **Fig. 3.24** shows the status

while the application is retrieving a sequence from GenBank based on a GI number. **Fig. 3.25** and **Fig. 3.26** show the status immediately after submitting a BLAST search and an intermediate stage before getting the results back. After all searches are complete, the system returns to the "`Ready`" status.

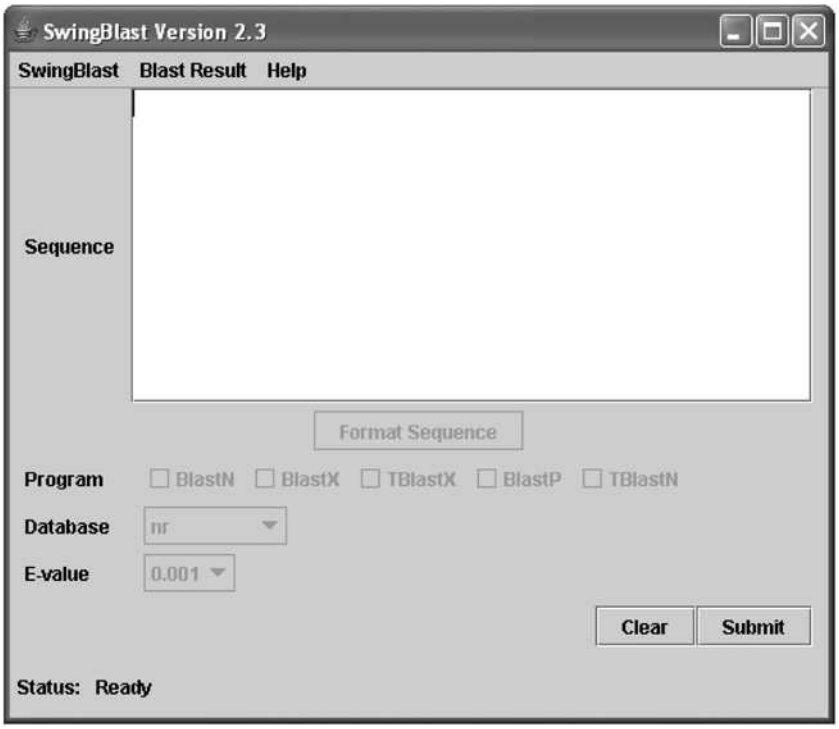

Fig. 3.23. Printing BLAST search status at start-up

Fig. 3.24. SwingBlast status during sequence retrieval from GenBank

Fig. 3.25. BLAST search status at the time of submission

Fig. 3.26. Printing BLAST search status before getting results

The code for adding BLAST search status is as follows. First, we add a status label at the bottom left side of the application:

```
      statusLabel = new JLabel(STATUS_LABEL);
      statusLabel.setPreferredSize(new Dimension(50, 30));
      statusText = new JLabel(STATUS_READY);
      JPanel statusPanel = new JPanel();

statusPanel.setBorder(BorderFactory.createEmptyBorder(0,    5,
5, 5));
      statusPanel.setLayout(new BorderLayout());
      statusPanel.add(statusLabel, BorderLayout.WEST);
      statusPanel.add(statusText, BorderLayout.CENTER);


  newContentPane.add(statusPanel, BorderLayout.SOUTH);
```

If a GI number has been entered in the sequence area, the application gets the corresponding sequence from GenBank and displays the

appropriate status message as shown in **Fig. 3.24**. The code for implementing this is shown below:

```
  if (isGenBankID) {
      boolean canGetSeq = true;
      final String statusText = "Retrieving sequence for
GenBank ID " + sequenceText;
      try {
          SwingUtilities.invokeAndWait(new Runnable() {
              public void run() {

SwingBlast2_3.this.statusText.setText(statusText);
              }
          });
          seqObject = genbankSequenceDB.getSequence(text);
      } catch (Exception e) {
          e.printStackTrace();
      }
      sequence = seqObject.seqString();
          SwingUtilities.invokeAndWait(new Runnable() {
              public void run() {
                  resetStatusText();
              }
          });
      } catch (IllegalArgumentException iae) {
          // ignore because we checked already!
      } catch (Exception e) {
      canGetSeq = false;
  }
  if (seq == null || seq.length() == 0 || !canGetSeq) {
      JOptionPane.showMessageDialog(SwingBlast2_3.this,
          "Cannot get the sequence for GenBank ID " +
sequenceText);
      return;
  }
```

Displaying BLAST Results Interactively

Finally, we will enhance the display capabilities of `SwingBlast` so we can view the results of the BLAST in a graphical and interactive manner. We will call this `SwingBlast` Version 2.4. The application will appear as shown in **Fig. 3.27**.

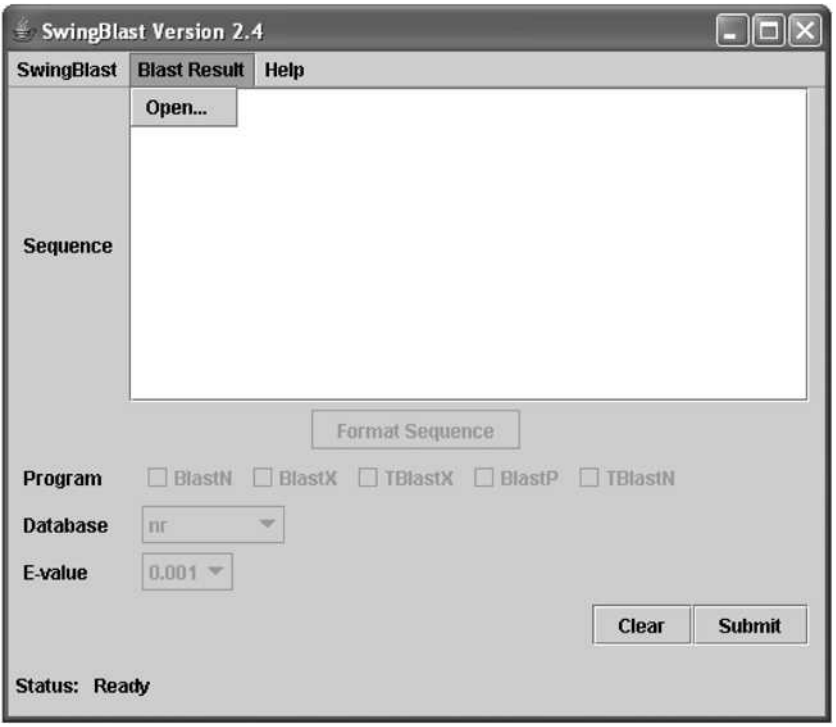

Fig. 3.27. Displaying BLAST results interactively

As seen in the **Fig. 3.27**, the user will select the `Blast` → `Open` menu button to access saved BLAST results. This will open a new window that will display the results in an interactive format. The data displayed in the graphical view is obtained by parsing information from the XML output shown in **Fig. 3.28.** The data parsed from the XML file includes such fields as the Hit_id, Hit_definition, Hit_accession, Hit_len, Hit_hsps, Hsp_number, Hsp_bit-score, Hsp_score, Hsp_evalue, etc. These fields describe the various attributes of a *High Scoring Sequence Pair* (HSP), such as the id, the definition, the GenBank accession number, the length, score, E-value etc. An HSP is a pair of aligned sequences of arbitrary but equal length, one derived from the query (input) sequence and one derived from the database it was searched against, that was returned by the BLAST search. The HSPs represent sequences whose alignment is locally maximal and for which the alignment score meets or exceeds a threshold or cutoff score provide by the user.

Fig. 3.28. XML file containing BLAST results data

BLAST results displayed in an interactive format are shown in **Fig. 3.29.**

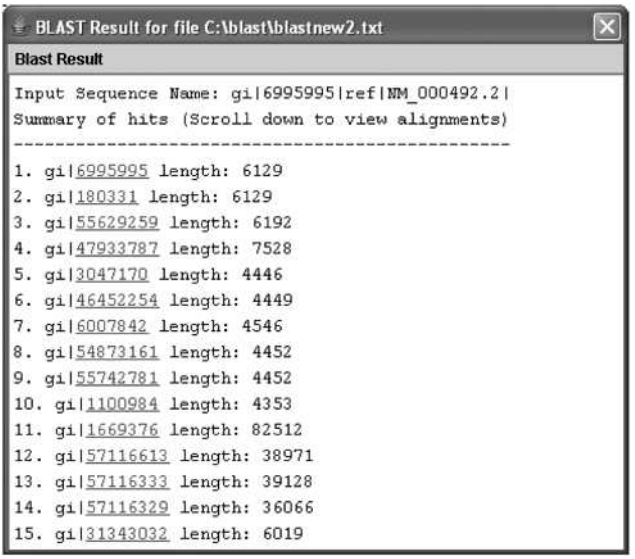

Fig. 3.29. Displaying BLAST results in an interactive format

Clicking on the GI number opens the GenBank record (**Fig. 3.30**).

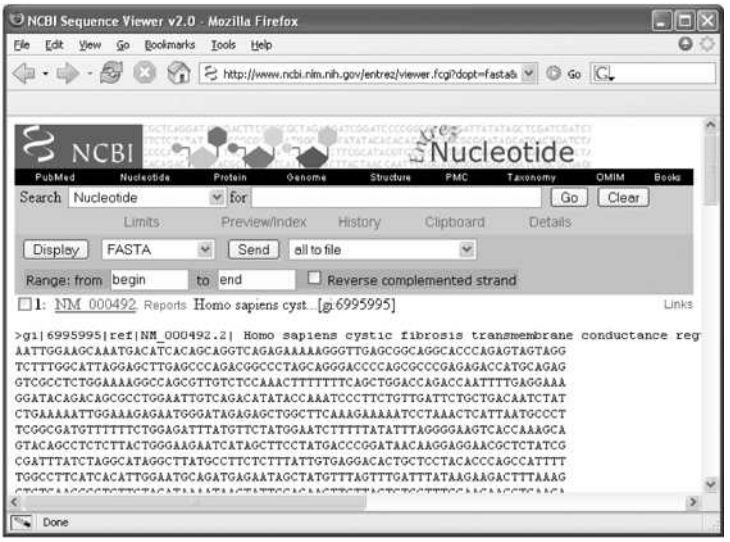

Fig. 3.30. Accessing GenBank record from BLAST results

Let's add the code that displays the BLAST results interactively. First, we add a menu item "BLAST Result" in the menu bar:

```
JMenu blastMenu = new JMenu("Blast Result");
openItem = new JMenuItem("Open...");
blastMenu.add(openItem);
menu.add(blastMenu);
```

Next we create a method called `displayBlastResult()` that takes a file name containing the BLAST results (that we saved earlier) as a parameter. The code is shown below:

```
    private void displayBlastResult(final String
blastFileName) {
    final JDialog blastDialog = new JDialog(this, "BLAST
Result for file " + blastFileName, false);
    final JTextArea textArea = new JTextArea();
    final Font sf = textArea.getFont();
    Font f = new Font("Monospaced", sf.getStyle(),
sf.getSize());
    textArea.setFont(f);

    Runnable runnable = new Runnable() {
      public void run() {
```

```
          Collection                blastHits                =
extractBlastHits(blastFileName);
          final String text = createReport(blastHits, new
ColorFormatterDNA());
          SwingUtilities.invokeLater(new Runnable() {
            public void run() {
              textArea.setText(text);
            }
          });
        }
      };
      new Thread(runnable).start();
      textArea.setLineWrap(true);
      final JMenuBar menuBar = new JMenuBar();
      JMenu menu = new JMenu("Blast Result");
      blastDialog.setJMenuBar(menuBar);
      JMenuItem openItem = new JMenuItem("Open...");
      openItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          final String blastResult = getBlastFileFromUser();
          if (blastResult != null)
            displayBlastResult(blastResult);
        }
      });
      menu.add(openItem);
      JMenuItem menuItem = new JMenuItem("Close");
      menuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          closeMenu(blastDialog);
        }
      });
      menu.add(menuItem);
      menuBar.add(menu);
      blastDialog.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          closeMenu(blastDialog);
        }
      });
      blastDialog.getContentPane().add(new
JScrollPane(textArea));
      blastDialog.setSize(CP_PREF_SIZE);
      centerLocation(blastDialog);
      blastDialog.setVisible(true);
    }
```

Next we add a method to create the report called `createReport()` which takes the *Collection* object and a color formatter object called `ColorFormatter`:

```
    private String createReport(Collection blastHits,
ColorFormatter colorFormatter) {
        StringBuffer summary = new
```

```
StringBuffer("<html><body  style=\"font-family:  'Monospaced',
Courier\">" +
                  "Input  Sequence  Name:  "  +  inputSeqName  +
"\n");
          StringBuffer alignments = null;
          if (blastHits == null || blastHits.size() == 0) {
              summary.append("No hits found from BLAST");
          } else {
              summary.append("Summary of hits (Scroll down to
view alignments)\n" +
                      "-------------------------------------
----------\n");
              BlastHit hit;
              BlastHsp hsp;
              Iterator iterator = blastHits.iterator();
              alignments  =  new  StringBuffer("\nAlignments\n"
+
                      "----------\n");
              int i = 1;
              while (iterator.hasNext()) {
                  hit = (BlastHit) iterator.next();
                  String hitId = hit.getHitId();
                  String genbankId = getGenBankId(hitId);
                  StringBuffer  tmp  =  new  StringBuffer(""  +
i++)
                          .append(".                        gi|<a
href=\"http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?dopt=fa
sta&list_uids="            +            genbankId            +
"\">").append(genbankId).append("</a>                  length:
").append(hit.getHitLen())
                          .append("\n");
                  summary.append(tmp);
                  alignments.append(tmp);
                  Iterator hspIte = hit.getHsps().iterator();
                  while (hspIte.hasNext()) {
                      hsp = (BlastHsp) hspIte.next();
                      alignments.append("Score                =
").append(hsp.getBitScore()).append(" bits  E-Value: ")

.append(hsp.getEvalue()).append("\n\n");
                      int              queryFrom             =
Integer.parseInt(hsp.getQueryStart());
                      int               queryTo              =
Integer.parseInt(hsp.getQueryEnd());
                      int            subjectFrom             =
Integer.parseInt(hsp.getSubjectStart());
                      int              subjecTo              =
Integer.parseInt(hsp.getSubjectEnd());
                      appendSequences(alignments,
hsp.getQseq(),        hsp.getMidline(),        hsp.getHseq(),
NUMB_OF_CHAR_PER_LINE,
                              queryFrom,              queryTo,
subjectFrom,  subjecTo,  queryFrom  <  queryTo,  subjectFrom  <
```

```
subjecTo, colorFormatter);
                        alignments.append("\n");
                    }
                    alignments.append("\n");
                }
                summary.append(alignments);
            }
            return
summary.append("</body></html>").toString().replaceAll("\n",
"<BR>");
        }
```

The color formatter that adds colors to the DNA alignment is as follows:

```
    private class ColorFormatter implements ColorFormatter {
       public String format(String s) {
          String upperCaseSeq = s.toUpperCase();
          String color;
          String letter;
          for (int i = 0; i < letters.length; i++) {
            letter = letters[i];
            color = letterToColor.get(letter);
            upperCaseSeq  =  upperCaseSeq.replaceAll(letter,
"<font color=\"" + color + "\">" + letter + "</font>");
          }
          return upperCaseSeq;
        }
     }
```

The output of the program, which we call SwingBlast 2.5, is shown in **Fig. 3.31-3.32**. **Fig. 3.1** shows a high scoring alignment with a top hit (no gaps) and **Fig. 3.32** shows alignment with a sequence with a lower score (with gaps).

Fig. 3.31. Alignment without gaps

Fig. 3.32. Alignment with gaps

Summary

In this Chapter we have demonstrated the development of a complete BLAST application using the NCBI QBlast package. We created BLAST API and demonstrated how they could be used for BLAST analysis using a user interface, which allows users to send sequences to the QBlast service. We demonstrated the use of existing BioJava libraries to retrieve sequences from GenBank. We also enhanced the BLAST search output by allowing users to link returned hits to GenBank and to view alignments in color. The NCBI BLAST service is an indispensable resource for biomedical research and is frequently among the first analytic tool that is used in routine research investigations. The purpose of this Chapter was to provide the user with a comprehensive understanding of the resource as well as to demonstrate how J2EE can be used to develop user-friendly applications to simplify this fundamental research activity. In the next

Chapter, we will explore another useful resource – PubMed and expose a different aspect of Java – namely, JavaServer Pages and Java Servlets.

Questions and Exercises

1. We have built SwingBlast to retrieve sequences from GenBank. Enhance the application by including the functionality to retrieve sequences from other data sources such as Ensembl, Swiss-Prot, etc.

2. The aim of BLAST searches is to provide information on the biological function of an unknown piece of nucleotide or protein sequence. Write an application that takes the basic SwingBlast framework and provides information on the returned hits from other functional data sources such as Entrez Gene, UniGene, Gene Expression Omnibus (GEO), HomoloGene, OMIM (Online Mendelian Inheritance in Man™), etc.

Additional Resources

- Ensembl - http://www.ensembl.org/index.html
- Entrez Gene - http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=gene
- GEO - http://www.ncbi.nlm.nih.gov/projects/geo/
- HomoloGene - http://www.ncbi.nlm.nih.gov/entrez/query.fcgi
- OMIM - http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=OMIM
- Swiss-Prot - http://www.expasy.org/sprot/
- UniGene - http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=unigene
- QBlast - http://www.ncbi.nlm.nih.gov/BLAST/Doc/urlapi.html

Selected Reading

UniGene: a unified view of the transcriptome. Pontius JU, Wagner L, Schuler GD. In: The NCBI Handbook. Bethesda (MD): National Center for Biotechnology Information; 2003.

NCBI GEO: mining millions of expression profiles - database and tools Tanya Barrett, Tugba O. Suzek, Dennis B. Troup, Stephen E. Wilhite, Wing-Chi Ngau, Pierre Ledoux, Dmitry Rudnev, Alex E. Lash, Wataru Fujibuchi and Ron Edgar. Nucleic Acids Research, 2005, Vol. 33, Database issue D562-D566.

An Overview of Ensembl. Ewan Birney, T. Daniel Andrews, Paul Bevan, Mario Caccamo, Yuan Chen, Laura Clarke, Guy Coates, James Cuff, Val Curwen, Tim Cutts, Thomas Down, Eduardo Eyras, Xose M. Fernandez-Suarez, Paul Gane, Brian Gibbins, James Gilbert, Martin Hammond, Hans-Rudolf Hotz, Vivek Iyer, Kerstin Jekosch, Andreas Kahari, Arek Kasprzyk, Damian Keefe, Stephen Keenan, Heikki Lehvaslaiho, Graham McVicker, Craig Melsopp, Patrick Meidl, Emmanuel Mongin, Roger Pettett, Simon Potter, Glenn Proctor, Mark Rae, Steve Searle, Guy Slater, Damian Smedley, James Smith, Will Spooner, Arne Stabenau, James Stalker, Roy Storey, Abel Ureta-Vidal, K. Cara Woodwark, Graham Cameron, Richard Durbin, Anthony Cox, Tim Hubbard, and Michele Clamp. Genome Res. 2004 May; 14(5):925-928.