BioJava In Anger

A Tutorial and Recipe Book for Those in a Hurry

Introduction:

BioJava can be both big and intimidating. For those of us who are in a hurry there really is a whole lot there to get your head around. This document is designed to help you develop BioJava programs that do 99% of common tasks without needing to read and understand 99% of the BioJava API.

The page was inspired by various programming cookbooks and follows a "How do I...?" type approach. Each How do I is linked to some example code that does what you want and sometimes more. Basically if you find the code you want and copy and paste it into your program you should be up and running quickly. I have endeavoured to over document the code to make it more obvious what I am doing so some of the code might look a bit bloated.

'BioJava in Anger' is maintained by Mark Schreiber. If you have any suggestions, questions or comments contact the biojava mailing list. To subscribe to this list go here

---

How Do I....?

Setup

- **Where do I get a Java installation?**
- **How do I get and install BioJava?**

Alphabets and Symbols

- **How do I get a DNA, RNA or Protein Alphabet?**
- **How do I make a custom Alphabet from custom Symbols?**
- **How do I make a CrossProductAlphabet such as a codon Alphabet?**
- **How do I break Symbols from CrossProduct Alphabets into their component Symbols?**
- **How can I tell if two Alphabets or Symbols are equal?**
- **How can I make an ambiguous Symbol like Y or R?**

Basic Sequence Manipulation

- **How do I make a Sequence from a String or make a Sequence Object back into a String?**

- How do I get a subsection of a Sequence?
- How do I transcribe a DNA Sequence to a RNA Sequence?
- How do I reverse complement a DNA or RNA Sequence?
- Sequences are immutable so how can I change it's name?

Translation

- How do I translate a DNA or RNA Sequence or SymbolList to Protein?
- How do I translate a single codon to a single amino acid?
- How do I use a non standard translation table?

Sequence I/O

- How do I write Sequences in Fasta format?
- How do I read in a Fasta file?
- How do I read a GenBank/EMBL/SwissProt file?
- How do I extract GenBank/EMBL/Swissprot sequences and write them as Fasta?
- How do I turn an ABI sequence trace into a BioJava Sequence?

Annotations

- How do I list the Annotations in a Sequence?
- How do I filter a Sequences based on their species (or another Annotation property)?

Locations and Features

- How do I specify a PointLocation?
- How do I specify a RangeLocation?
- How do CircularLocations work?
- How can I make a Feature?
- How can I filter Features by type?

BLAST and FASTA

- How do I set up a BLAST parser?
- How do I set up a FASTA parser?
- How do I extract information from parsed results?

Counts and Distributions

- How do I count the residues in a Sequence?
- How do I calculate the frequency of a Symbol in a Sequence?

- How can I turn a Count into a Distribution?
- How can I generate a random sequence from a Distribution?
- How can I find the amount of information or entropy in a Distribution?
- What is an easy way to tell if two Distributions have equal weights?
- How can I make an OrderNDistribution over a custom Alphabet?
- How can I write a Distribution as XML?

Weight Matrices and Dynamic Programming

- How do I use a WeightMatrix to find a motif?
- How do I make a HMMER like profile HMM?

User Interfaces

- How can I visualize Annotations and Features as a tree?
- How can I display a Sequence in a GUI?
- How do I display Sequence coordinates?
- How can I display features?

Disclaimer:

This code is generously donated by people who probably have better things to do. Where possible we test it but errors may have crept in. As such, all code and advice here in has no warranty or guarantee of any sort. You didn't pay for it and if you use it we are not responsible for anything that goes wrong. Be a good programmer and test it yourself before unleashing it on your corporate database.

This code is open-source. A good definition of open-source can be found here. If you agree with that definition then you can use it.

How do I get a DNA, RNA or Protein Alphabet?

In BioJava Alphabets are collections of Symbols. Common biological alphabets (DNA, RNA, protein etc) are registered with the BioJava AlphabetManager at startup and can be accessed by name. The DNA, RNA and protein alphabets can also be accessed using convenient static methods from DNATools, RNATools and ProteinTools respectively.

Both of these approaches are shown in the example below

```
import org.biojava.bio.symbol.*;
import java.util.*;
import org.biojava.bio.seq.*;

public class AlphabetExample {
  public static void main(String[] args) {
    Alphabet dna, rna, prot;

    //get the DNA alphabet by name
    dna = AlphabetManager.alphabetForName("DNA");

    //get the RNA alphabet by name
    rna = AlphabetManager.alphabetForName("RNA");

    //get the Protein alphabet by name
    prot = AlphabetManager.alphabetForName("PROTEIN");
    //get the protein alphabet that includes the * termination Symbol
    prot = AlphabetManager.alphabetForName("PROTEIN-TERM");

    //get those same Alphabets from the Tools classes
    dna = DNATools.getDNA();
    rna = RNATools.getRNA();
    prot = ProteinTools.getAlphabet();
    //or the one with the * symbol
    prot = ProteinTools.getTAlphabet();

  }
}
```

How do I make a custom Alphabet from custom Symbols?

This example demonstrates the creation of a 'binary' alphabet that will have two Symbols, zero and one. The custom made Symbols and Alphabet can then be used to make SymbolLists, Sequences, Distributions etc.

```
import org.biojava.bio.symbol.*;
import org.biojava.bio.*;
import java.util.*;

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

    //make the "zero" Symbol with no annotation
    Symbol zero =
        AlphabetManager.createSymbol("zero", Annotation.EMPTY_ANNOTATION);

    //make the "one" Symbol
    Symbol one =
        AlphabetManager.createSymbol("one", Annotation.EMPTY_ANNOTATION);

    //collect the Symbols in a Set
    Set symbols = new HashSet();
    symbols.add(zero); symbols.add(one);

    //make the Binary Alphabet
    FiniteAlphabet binary = new SimpleAlphabet(symbols, "Binary");

    //iterate through the symbols to show everything works
    for (Iterator i = binary.iterator(); i.hasNext(); ) {
      Symbol sym = (Symbol)i.next();
      System.out.println(sym.getName());
    }

    //it is usual to register newly creates Alphabets with the AlphabetManager
    AlphabetManager.registerAlphabet(binary.getName(), binary);

    /*
     * The newly created Alphabet will have been registered with the
     * AlphabetManager under the name "Binary". If you retreive an instance
     * of it using this name it should be canonical with the previous instance
     */
    Alphabet alpha = AlphabetManager.alphabetForName("Binary");
```

```
    //check canonical status
    System.out.println(alpha == binary);
  }
}
```

How do I make a CrossProductAlphabet such as a codon Alphabet

CrossProductAlphabets result from the multiplication of other Alphabets. CrossProductAlphabets are used to wrap up 2 or more Symbols into a single "cross product" Symbol. For example using a 3 way cross of the DNA alphabet you could wrap a codon as a Symbol. You could then count those codon Symbols in a Count or you could used them in a Distribution.

CrossProductAlphabets can be created by name (if the component Alphabets are registered in the AlphabetManager) or by making a list of the desired Alphabets and creating the Alphabet from the List. Both approaches are shown in the example below.

```
import java.util.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

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

    //make a CrossProductAlphabet from a List
    List l = Collections.nCopies(3, DNATools.getDNA());
    Alphabet codon = AlphabetManager.getCrossProductAlphabet(l);

    //get the same Alphabet by name
    Alphabet codon2 =
        AlphabetManager.generateCrossProductAlphaFromName("(DNA x DNA x DNA)");

    //show that the two Alphabets are canonical
    System.out.println(codon == codon2);
  }
}
```

How do I break Symbols from CrossProductAlphabets into their component Symbols?

CrossProductAlphabets are used to represent groups of Symbols as a single Symbol. This is very useful for treating things like codons as single Symbols. Sometimes however, you might want to covert the symbols back into their component Symbols. The following recipe demonstrates how this can be done.

The Symbols from a CrossProductAlphabet are implementations of the AtomicSymbol interface. The prefix 'Atomic' suggests that the Symbol cannot be divided so one might ask, 'how can an indivisible Symbol be divided into it's component parts?'. The full definition of the AtomicSymbol is that it cannot be divided into a simpler Symbol that is still part of the same Alphabet. The component parts of an AtomicSymbol from a CrossProductAlphabet are not members of the same Alphabet so the 'Atomic' definition still stands. A codon would be from the (DNA x DNA x DNA) Alphabet whereas the components of the codon Symbol are from the DNA alphabet.

Contrast this with the definition of a BasisSymbol. A BasisSymbol can be validly divided into components that are still part of the same Alphabet. In this way a BasisSymbol can be ambiguous. For further discussion of BasisSymbols follow this link.

```
package biojava_in_anger;

import java.util.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

public class BreakingComponents {
  public static void main(String[] args) {
    //make the 'codon' alphabet
    List l = Collections.nCopies(3, DNATools.getDNA());
    Alphabet alpha = AlphabetManager.getCrossProductAlphabet(l);

    //get the first symbol in the alphabet
    Iterator iter = ((FiniteAlphabet)alpha).iterator();
    AtomicSymbol codon = (AtomicSymbol)iter.next();
```

```
    System.out.print(codon.getName()+" is made of: ");

    //break it into a list its components
    List symbols = codon.getSymbols();
    for(int i = 0; i &LT; symbols.size(); i++){
      if(i != 0)
        System.out.print(", ");
      Symbol sym = (Symbol)symbols.get(i);
      System.out.print(sym.getName());
    }
  }
}
```

How can I tell if two Symbols or Alphabets are equal?

In Biojava the same Alphabets and the same Symbols are canonical no matter how they where constructed or where they came from. This means that if two DNA alphabets (or Symbols from those alphabets) are instantiated at different times are equal via both the .equals() and == functions. Also Symbols from the PROTEIN and the PROTEIN-TERM alphabets are canonical as are Symbols from the IntegerAlphabet and the SubIntegerAlphabets.

This is even true of Alphabets and Symbols on different virtual machines (thanks to some Serialization magic) which means BioJava works across RMI.

```
import org.biojava.bio.symbol.*;
import org.biojava.bio.seq.*;

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

    //get the DNA alphabet two ways
    Alphabet a1 = DNATools.getDNA();
    Alphabet a2 = AlphabetManager.alphabetForName("DNA");

    //are they equal
    System.out.println("equal: "+ a1.equals(a2));
    //are they canonical
    System.out.println("canonical: "+ (a1 == a2));
  }
}
```

How can I make an ambiguous Symbol like Y or R?

The IBU defines standard codes for symbols that are ambiguous such as Y to indicate C or T and R to indicate G or C or N to indicate any nucleotide. BioJava represents these Symbols as BasisSymbols. BasisSymbol objects can contain one or more component Symbols that are valid members of the same Alphabet as the BasisSymbol and are therefore capable of being ambiguous.

Generally an ambiguity Symbol is retrieved by calling the getAmbiguity(Set symbols) method from the Alphabet that the Symbol is intended to come from. In the case of making the Symbol Y the set 'symbols' used as an argument will contain the DNA Alphabet Symbols 'C' and 'T'.

```
import org.biojava.bio.symbol.*;
import org.biojava.bio.seq.*;
import java.util.*;

public class Ambiguity {
  public static void main(String[] args) {
    try {
      //get the DNA Alphabet
      Alphabet dna = DNATools.getDNA();

      //make the 'Y' symbol
      Set symbolsThatMakeY = new HashSet();
      symbolsThatMakeY.add(DNATools.c());
      symbolsThatMakeY.add(DNATools.t());
      Symbol y = dna.getAmbiguity(symbolsThatMakeY);

      //print information about 'Y' basis Symbol
      System.out.println("Formal name of 'Y' is: "+y.getName());
      System.out.println("Class type of 'Y' is: "+y.getClass().getName());

      //break the Y BasisSymbol into its component AtomicSymbols
      Alphabet matches = y.getMatches();
      System.out.print("The 'Y' Symbol is made of: ");

      //we know that there will be a finite set of matches so its ok to cast it
      for(Iterator i = ((FiniteAlphabet)matches).iterator(); i.hasNext();){
        Symbol sym = (Symbol)i.next();
        System.out.print(sym.getName());
        if(i.hasNext())
          System.out.print(", ");
      }

    }
    catch (IllegalSymbolException ex) {
      ex.printStackTrace();
```

How can I make an ambiguous Symbol like Y or R?

```
      }
    }
}
```

How do I make a Sequence from a String or make a Sequence Object back into a String?

A lot of the time we see sequence represented as a String of characters eg "atgccgtggcatcgaggcatatagc". It's a convenient method for viewing and succinctly representing a more complex biological polymer. BioJava makes use of SymbolLists and Sequences to represent these biological polyners as Objects. Sequences extend SymbolLists and provide extra methods to store things like the name of the sequence and any features it might have but you can think of a Sequence as a SymbolList.

Within Sequence and SymbolList the polymer is *not* stored as a String. BioJava differentiates different polymer residues using Symbol objects that come from different Alphabets. In this way it is easy to tell if a sequence is DNA or RNA or something else and the 'A' symbol from DNA is not equal to the 'A' symbol from RNA. The details of Symbols, SymbolLists and Alphabets are covered here. The crucial part is there needs to be a way for a programmer to convert between the easily readable String and the BioJava Object and the reverse. To do this BioJava has Tokenizers that can read a String of text and parse it into a BioJava Sequence or SymbolList object. In the case of DNA, RNA and Protein you can do this with a single method call. The call is made to a static method from either DNATools, RNATools or ProteinTools.

String to SymbolList

```
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

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

    try {
      //create a DNA SymbolList from a String
      SymbolList dna = DNATools.createDNA("atcggtcggctta");

      //create a RNA SymbolList from a String
      SymbolList rna = RNATools.createRNA("auugccuacauaggc");

      //create a Protein SymbolList from a String
      SymbolList aa = ProteinTools.createProtein("AGFAVENDSA");
    }
```

```
    catch (IllegalSymbolException ex) {
      //this will happen if you use a character in one of your strings that is
      //not an accepted IUB Character for that Symbol.
      ex.printStackTrace();
    }

  }
}
```

String to Sequence

```
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

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

    try {
      //create a DNA sequence with the name dna_1
      Sequence dna = DNATools.createDNASequence("atgctg", "dna_1");

      //create an RNA sequence with the name rna_1
      Sequence rna = RNATools.createRNASequence("augcug", "rna_1");

      //create a Protein sequence with the name prot_1
      Sequence prot = ProteinTools.createProteinSequence("AFHS", "prot_1");
    }
    catch (IllegalSymbolException ex) {
      //an exception is thrown if you use a non IUB symbol
      ex.printStackTrace();
    }
  }
}
```

SymbolList to String

You can call the seqString() method on either a SymbolList or a Sequence to get it's Stringified version.

```
import org.biojava.bio.symbol.*;

public class SymbolListToString {
  public static void main(String[] args) {
    SymbolList sl = null;
    //code here to instantiate sl

    //convert sl into a String
    String s = sl.seqString();
  }
}
```

How do I get a subsection of a Sequence?

Given a Sequence object we might only be interested in examining the first 10 bases or we might want to get a region between two points. You might also want to print a subsequence to an OutputStream like STDOUT how could you do this?

BioJava uses a biological coordinate system for identifying bases. The first base is numbered 1 and the last base index is equal to the length of the sequence. Note that this is different from String indexing which starts at 0 and proceedes to length -1. If you attempt to access a region outside of 1...length an IndexOutOfBoundsException will occur.

Getting a Sub - Sequence

```
SymbolList symL = null;

//code here to generate a SymbolList

//get the first Symbol
Symbol sym = symL.symbolAt(1);

//get the first three bases
SymbolList symL2 = symL.subList(1,3);

//get the last three bases
SymbolList symL3 = symL.subList(symL.length() - 3, symL.length());
```

Printing a Sub - Sequence

```
//print the last three bases of a SymbolList or Sequence
String s = symL.subStr(symL.length() - 3, symL.length());
System.out.println(s);
```

Complete Listing

```
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

public class SubSequencing {
  public static void main(String[] args) {
    SymbolList symL = null;

    //generate an RNA SymbolList
    try {
      symL = RNATools.createRNA("auggcaccguccagauu");
    }
```

```
    catch (IllegalSymbolException ex) {
      ex.printStackTrace();
    }

    //get the first Symbol
    Symbol sym = symL.symbolAt(1);

    //get the first three bases
    SymbolList symL2 = symL.subList(1,3);

    //get the last three bases
    SymbolList symL3 = symL.subList(symL.length() - 3, symL.length());

    //print the last three bases
    String s = symL.subStr(symL.length() - 3, symL.length());
    System.out.println(s);
  }
}
```

How do I Transcribe a DNA Sequence to an RNA Sequence?

In BioJava DNA and RNA Sequences and SymbolLists are made using different Alphabets you can convert from DNA to RNA using the static method transcribe() in RNATools.

```
import org.biojava.bio.symbol.*;
import org.biojava.bio.seq.*;

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

    try {
      //make a DNA SymbolList
      SymbolList symL = DNATools.createDNA("atgccgaatcgtaa");

      //transcribe it to RNA
      symL = RNATools.transcribe(symL);

      //just to prove it worked
      System.out.println(symL.seqString());

    }
    catch (IllegalSymbolException ex) {
      //this will happen if you try and make the DNA seq using non IUB symbols
      ex.printStackTrace();
    }catch (IllegalAlphabetException ex) {
      //this will happen if you try and transcribe a non DNA SymbolList
      ex.printStackTrace();
    }
  }
}
```

How do I Reverse Complement a Sequence or SymbolList?

To reverse complement a DNA SymbolList or Sequence simply use the DNATool.reverseComplement(SymbolList sl) method. An equivalent method is found in RNATools for performing the same operation on RNA based Sequences and SymbolLists.

```
import org.biojava.bio.symbol.*;
import org.biojava.bio.seq.*;

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

    try {
      //make a DNA SymbolList
      SymbolList symL = DNATools.createDNA("atgcacgggaactaa");

      //reverse complement it
      symL = DNATools.reverseComplement(symL);

      //prove that it worked
      System.out.println(symL.seqString());
    }
    catch (IllegalSymbolException ex) {
      //this will happen if you try and make the DNA seq using non IUB symbols
      ex.printStackTrace();
    }catch (IllegalAlphabetException ex) {
      //this will happen if you try and reverse complement a non DNA sequence
using DNATools
      ex.printStackTrace();
    }
  }
```

How can I change a Sequence's name?

Mostly BioJava Sequence objects are immutable. This is really a safety feature to prevent changes corrupting the integrity of the data. A consequence of this is that there is no setName() method in Sequence. One way to change your "view" of a Sequence is to make a ViewSequence using the original Sequence as an argument in the constructor. Behind the scenes the ViewSequence wrapper intercepts some of the method calls to the underlying Sequence which gives the possibility of changing the name.

The following program demonstrates this.

```
import java.io.*;


import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;
import org.biojava.bio.symbol.*;

public class NameChange {
  public static void main(String[] args) {
    try {
      Sequence seq =
          DNATools.createDNASequence("atgcgctaggctag","gi|12356|ABC123");

      //create a veiw on the sequence and change its name
      ViewSequence seq2 = new ViewSequence(seq, "ABC123");

      //print to FASTA to prove the name has changed
      SeqIOTools.writeFasta(System.out, seq2);
    }
    catch (IllegalSymbolException ex) {
      //tried to make seq with non DNA symbol
      ex.printStackTrace();
    }catch (IOException ex) {
      //couldn't print seq2 to System out??
      ex.printStackTrace();
    }
  }
}
```

How Do I Translate a SymbolList or Sequence?

To translate a DNA sequence you need to do the following

- **Transcribe to RNA.**
- **Get a triplet (codon) view on the SymbolList.**
- **Translate to protein.**

Almost all of this can be achieved using static methods from BioJava tools classes. The following block of code demonstrates the procedure. Obviously if you already have an RNA sequence there is no need to transcribe it.

NOTE: if you try and create a 'triplet view' on a SymbolList or Sequence who's length is not evenly divisible by three an IllegalArgumentException will be thrown. See 'how to get a subsequence' for a description of how to get a portion of a Sequence for translation.

```
import org.biojava.bio.symbol.*;
import org.biojava.bio.seq.*;

public class Translate {

  public static void main(String[] args) {
    try {
      //create a DNA SymbolList
      SymbolList symL = DNATools.createDNA("atggccattgaatga");

      //transcribe to RNA
      symL = RNATools.transcribe(symL);

      //translate to protein
      symL = RNATools.translate(symL);

      //prove that it worked
      System.out.println(symL.seqString());
    }
    catch (IllegalAlphabetException ex) {

      /*
       * this will occur if you try and transcribe a non DNA sequence or translate
       * a sequence that isn't a triplet view on a RNA sequence.
       */
      ex.printStackTrace();
    }catch (IllegalSymbolException ex) {
      // this will happen if non IUB characters are used to create the DNA
SymbolList
      ex.printStackTrace();
```

```
    }
}
```

How do I translate a single codon to a single amino acid?

The general translation example shows how to use RNATools to translate a RNA SymbolList into a Protein SymbolList but most of what goes on is hidden behind the convenience method translate(). If you only want to translate a single codon into a single amino acid you get exposed to a bit more of the gory detail but you also get a chance to figure out more of what is going on under the hood.

There are actually a number of ways to do this, below I have presented only one.

```
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

public class SingleTranslationDemo {
  public static void main(String[] args) {
    //make a compound alphabet where codons are Symbols
    Alphabet a = AlphabetManager.alphabetForName("(RNA x RNA x RNA)");

    //get our translation table using one of the static names from TranslationTable
    TranslationTable table = RNATools.getGeneticCode(TranslationTable.UNIVERSAL);

    try {
      //make a 'codon'
      SymbolList codon = RNATools.createRNA("UUG");

      //get the representation of that codon as a Symbol
      Symbol sym = a.getSymbol(codon.toList());

      //translate to amino acid
      Symbol aminoAcid = table.translate(sym);

      /*
       * This bit is not required for the translation it just proves that the
       * Symbol is from the right Alphabet. An Exception will be thrown if it
       * isn't.
       */
      ProteinTools.getTAlphabet().validate(aminoAcid);
    }
    catch (IllegalSymbolException ex) {
      ex.printStackTrace();
    }
  }
}
```

How do I use a non standard translation table?

The convenient translate() method in RNATools, used in the general translation example, is only useful if you want to use the "Universal" translation table. This is not so good if you want to use one of those weird Mitochondrial translation tables. Fortunately this can be done in BioJava. RNATools also has a static method getGeneticCode(String name) that lets you get a TranslationTable by name.

The following TranslationTables are available:

FLATWORM_MITOCHONDRIAL
YEAST_MITOCHONDRIAL
ASCIDIAN_MITOCHONDRIAL
EUPLOTID_NUCLEAR
UNIVERSAL
INVERTEBRATE_MITOCHONDRIAL
BLEPHARISMA_MACRONUCLEAR
ALTERNATIVE_YEAST_NUCLEAR
BACTERIAL
VERTEBRATE_MITOCHONDRIAL
CILIATE_NUCLEAR
MOLD_MITOCHONDRIAL
ECHINODERM_MITOCHONDRIAL

These are also the valid names that can be used as an argument in the static RNATools.getGeneticCode(String name) method. These names are also available as static Strings in the TranslationTools class.

The following program shows the use of the Euplotid Nuclear translation table (where UGA = Cys).

```
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

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

    //get the Euplotoid translation table
    TranslationTable eup = RNATools.getGeneticCode(TranslationTable.EUPL_NUC);

    try {
      //make a DNA sequence including the 'tga' codon
      SymbolList seq = DNATools.createDNA("atgggcccatgaaaaggcttggagtaa");

      //transcribe to RNA
      seq = RNATools.transcribe(seq);

      //veiw the RNA sequence as codons, this is done internally by
RNATool.translate()
```

```
      seq = SymbolListViews.windowedSymbolList(seq, 3);

      //translate
      SymbolList protein = SymbolListViews.translate(seq, eup);

      //print out the protein
      System.out.println(protein.seqString());
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

How Do I Print A Sequence in Fasta Format?

FASTA format is a fairly standard bioinformatics output that is convenient and easy to read. BioJava has a tools class called SeqIOTools that provides static convenience methods to perform a number of common bioinformatics IO tasks. The follwing snippets demonstrate how to print a Sequence or even a whole SequenceDB in FASTA format to an OutputStream like System.out. All of the WriteXX methods from SeqIOTools take an OutputStream as an argument. In this way you can pipe the newly formatted sequence to a file or another method or STDOUT, STDERR etc.

SeqIOTools is in the package org.biojava.bio.seq.io

Printing a SequenceDB

```
//make a instance of the SequenceDB interface
SequenceDB db = new HashSequenceDB();

//add the sequences to the DB
db.addSequence(seq1);
db.addSequence(seq2);

/*
 * now print it to an output stream in FASTA format using a static method
 * from the utility class SeqIOTools. In this case our output stream is
 * STDOUT
 */
SeqIOTools.writeFasta(System.out, db);
```

Printing from a SequenceIterator

Many readXXX() methods from SeqIOTools return a SequenceIterator that iterates over all the Sequences in a file. Most of teh writeXXX() methods from SeqIOTools have a version of the methods that takes a SequenceIterator as and argument eg.

```
SequenceIterator iter =
    (SequenceIterator)SeqIOTools.fileToBiojava(fileType, br);

//and now write it all to FASTA, (you can write to any OutputStream, not
just System.out)
SeqIOTools.writeFasta(System.out, iter);
```

Printing a Single Sequence

```
/*
 * SeqIOTools also has a method that takes a single sequence so you don't
 * have to make a SequenceDB
 */
SeqIOTools.writeFasta(System.out, seq1);
```

How do I read Sequences from a Fasta File?

One of the most frequent I/O tasks is the reading of a flat file representation of sequence into memory. SeqIOTools provides some basic static methods to read files into BioJava. There is actually more than one solution. The more specific is demonstrated first and the more general second.

Solution 1

```
import java.io.*;
import java.util.*;


import org.biojava.bio.*;
import org.biojava.bio.seq.db.*;
import org.biojava.bio.seq.io.*;
import org.biojava.bio.symbol.*;

public class ReadFasta {

  /**
   * The programs takes two args the first is the file name of the Fasta file.
   * The second is the name of the Alphabet. Acceptable names are DNA RNA or
PROTEIN.
   */
  public static void main(String[] args) {

    try {
      //setup file input
      String filename = args[0];
      BufferedInputStream is =
          new BufferedInputStream(new FileInputStream(filename));


      //get the appropriate Alphabet
      Alphabet alpha = AlphabetManager.alphabetForName(args[1]);

      //get a SequenceDB of all sequences in the file
      SequenceDB db = SeqIOTools.readFasta(is, alpha);
    }
    catch (BioException ex) {
      //not in fasta format or wrong alphabet
      ex.printStackTrace();
    }catch (NoSuchElementException ex) {
      //no fasta sequences in the file
      ex.printStackTrace();
    }catch (FileNotFoundException ex) {
```

```
      //problem reading file
      ex.printStackTrace();
    }
  }
```

Solution 2

```
import org.biojava.bio.seq.io.*;
import org.biojava.bio.seq.*;
import java.io.*;

public class ReadFasta2 {

  /**
   * This program will read any file supported by SeqIOTools it takes two
   * arguments, the first is the file name the second is the int constant
   * for the file type in SeqIOTools. See SeqIOTools for possible file types.
   * The constant for Fasta DNA is 1
   *
   */
  public static void main(String[] args) {
    try {
      //prepare a BufferedReader for file io
      BufferedReader br = new BufferedReader(new FileReader(args[0]));

      //get the int constant for Fasta file type
      int fileType = Integer.parseInt(args[1]);

      /*
       * get a Sequence Iterator over all the sequences in the file.
       * SeqIOTools.fileToBiojava() returns an Object. If the file read
       * is an alignment format like MSF and Alignment object is returned
       * otherwise a SequenceIterator is returned.
       */
      SequenceIterator iter =
          (SequenceIterator)SeqIOTools.fileToBiojava(fileType, br);
    }
    catch (FileNotFoundException ex) {
      //can't find file specified by args[0]
      ex.printStackTrace();
    }catch (NumberFormatException ex) {
      //args[1] is not an integer
      ex.printStackTrace();
    }
  }
}
```

How Do I read a GenBank, SwissProt or EMBL file?

The SeqIOTools class contains methods for reading GenBank, SwissProt and EMBL files. Because any of these files can contain more than one sequence entry SeqIOTools will return a SequenceIterator which can be used to iterate through the individual sequences. One of the attractive features of this model is that the Sequences are only parsed and created as needed so very large collections of sequences can be handled with moderate resources.

Information in the file is store in the Sequence as Annotations or where there is location information as Features.

Three specific solutions are presented (which are all very similar) followed by a generic solution (for biojava1.3 pre1). A fourth solution revises the generic solution for the biojava1.3 API which is a bit friendlier.

Reading GenBank

```
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;
import java.io.*;
import org.biojava.bio.*;
import java.util.*;

public class ReadGB {
  public static void main(String[] args) {
    BufferedReader br = null;

    try {

      //create a buffered reader to read the sequence file specified by args[0]
      br = new BufferedReader(new FileReader(args[0]));

    }
    catch (FileNotFoundException ex) {
      //can't find the file specified by args[0]
      ex.printStackTrace();
      System.exit(-1);
    }

    //read the GenBank File
    SequenceIterator sequences = SeqIOTools.readGenbank(br);

    //iterate through the sequences
    while(sequences.hasNext()){
      try {
```

```
        Sequence seq = sequences.nextSequence();
        //do stuff with the sequence

      }
      catch (BioException ex) {
        //not in GenBank format
        ex.printStackTrace();
      }catch (NoSuchElementException ex) {
        //request for more sequence when there isn't any
        ex.printStackTrace();
      }
    }
  }
}
```

Reading SwissProt

```
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;
import java.io.*;
import org.biojava.bio.*;
import java.util.*;

public class ReadSwiss {
  public static void main(String[] args) {
    BufferedReader br = null;

    try {

      //create a buffered reader to read the sequence file specified by args[0]
      br = new BufferedReader(new FileReader(args[0]));

    }
    catch (FileNotFoundException ex) {
      //can't find the file specified by args[0]
      ex.printStackTrace();
      System.exit(-1);
    }

    //read the SwissProt File
    SequenceIterator sequences = SeqIOTools.readSwissprot(br);

    //iterate through the sequences
    while(sequences.hasNext()){
      try {

        Sequence seq = sequences.nextSequence();
        //do stuff with the sequence
```

```
      }
      catch (BioException ex) {
        //not in SwissProt format
        ex.printStackTrace();
      }catch (NoSuchElementException ex) {
        //request for more sequence when there isn't any
        ex.printStackTrace();
      }
    }
  }
}
```

Reading EMBL

```
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;
import java.io.*;
import org.biojava.bio.*;
import java.util.*;

public class ReadEMBL {
  public static void main(String[] args) {
    BufferedReader br = null;

    try {

      //create a buffered reader to read the sequence file specified by args[0]
      br = new BufferedReader(new FileReader(args[0]));

    }
    catch (FileNotFoundException ex) {
      //can't find the file specified by args[0]
      ex.printStackTrace();
      System.exit(-1);
    }

    //read the EMBL File
    SequenceIterator sequences = SeqIOTools.readEmbl(br);

    //iterate through the sequences
    while(sequences.hasNext()){
      try {

        Sequence seq = sequences.nextSequence();
        //do stuff with the sequence

      }
      catch (BioException ex) {
        //not in EMBL format
```

```
        ex.printStackTrace();
      }catch (NoSuchElementException ex) {
        //request for more sequence when there isn't any
        ex.printStackTrace();
      }
    }
  }
}
```

GeneralReader (biojava 1.3 pre 1)

```
import org.biojava.bio.seq.io.*;
import org.biojava.bio.seq.*;
import java.io.*;

public class GeneralReader {

  /**
   * This program will read any file supported by SeqIOTools it takes two
   * arguments, the first is the file name the second is the int constant
   * for the file type in SeqIOTools. See SeqIOTools for possible file types.
   * The constants used are:
   * UNKNOWN = 0;
   * FASTADNA = 1;
   * FASTAPROTEIN = 2;
   * EMBL = 3;
   * GENBANK = 4;
   * SWISSPROT = 5;
   * GENPEPT = 6;
   * MSFDNA = 7;
   * FASTAALIGNDNA = 9;
   * MSFPROTEIN = 10;
   * FASTAALIGNPROTEIN = 11;
   * MSF = 12;               //only appropriate for reading
   *
   */
  public static void main(String[] args) {
    try {
      //prepare a BufferedReader for file io
      BufferedReader br = new BufferedReader(new FileReader(args[0]));

      //get the int constant for the file type
      int fileType = Integer.parseInt(args[1]);

      /*
       * get a Sequence Iterator over all the sequences in the file.
       * SeqIOTools.fileToBiojava() returns an Object. If the file read
       * is an alignment format like MSF and Alignment object is returned
       * otherwise a SequenceIterator is returned.
```

```
       */
      SequenceIterator iter =
          (SequenceIterator)SeqIOTools.fileToBiojava(fileType, br);
    }
    catch (FileNotFoundException ex) {
      //can't find file specified by args[0]
      ex.printStackTrace();
    }catch (NumberFormatException ex) {
      //args[1] is not an integer
      ex.printStackTrace();
    }
  }
}
```

GeneralReader (biojava 1.3)

```
import java.io.*;


import org.biojava.bio.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;

public class GeneralReader {

  /**
   * This program will read any file supported by SeqIOTools it takes three
   * arguments, the first is the file name the second is the format type the
   * third is the type of residue being read. Illegal combinations such as
   * SwissProt and DNA will cause an exception.
   *
   * Allowed formats are: (case insensitive).
   *
   * FASTA
   * EMBL
   * GENBANK
   * SWISSPROT (or swiss)
   * GENPEPT
   *
   * Allowed sequence types are: (case insensititve).
   *
   * DNA
   * AA (or Protein)
   * RNA
   *
   */
  public static void main(String[] args) {
    try {
      //prepare a BufferedReader for file io
      BufferedReader br = new BufferedReader(new FileReader(args[0]));
```

```
      //the flat file format
      String format = args[1];

      //the Alphabet
      String alpha = args[2];

      //get the int value for the format and alphabet


      /*
       * get a Sequence Iterator over all the sequences in the file.
       * SeqIOTools.fileToBiojava() returns an Object. If the file read
       * is an alignment format like MSF and Alignment object is returned
       * otherwise a SequenceIterator is returned.
       */
      SequenceIterator iter =
          (SequenceIterator)SeqIOTools.fileToBiojava(format, alpha, br);

      // do something with the sequences
      SeqIOTools.writeFasta(System.out, iter);
    }
    catch (FileNotFoundException ex) {
      //can't find file specified by args[0]
      ex.printStackTrace();
    }catch (BioException ex) {
      //invalid file format name
      ex.printStackTrace();
    }catch (IOException ex){
      //error writing to fasta
      ex.printStackTrace();
    }
  }
}
```

How do I extract Sequences from GenBank/ EMBL/ SwissProt etc and write them as Fasta?

To perform this task we are going to extend the general reader from the previous demo and include in it the ability to write sequence data in fasta format. Two examples are provided, one uses the slightly earlier biojava1.3 pre1 and the second uses the more up to date biojava 1.3

(Using Biojava 1.3 pre 1)

```
import org.biojava.bio.seq.io.*;
import org.biojava.bio.seq.*;
import java.io.*;

public class WriteToFasta {

  /**
   * This program will read any file supported by SeqIOTools it takes two
   * arguments, the first is the file name the second is the int constant
   * for the file type in SeqIOTools. See SeqIOTools for possible file types.
   * The constants used are:
   * UNKNOWN = 0;
   * FASTADNA = 1;
   * FASTAPROTEIN = 2;
   * EMBL = 3;
   * GENBANK = 4;
   * SWISSPROT = 5;
   * GENPEPT = 6;
   * MSFDNA = 7;
   * FASTAALIGNDNA = 9;
   * MSFPROTEIN = 10;
   * FASTAALIGNPROTEIN = 11;
   * MSF = 12;
   *
   */
  public static void main(String[] args) {
    try {
      //prepare a BufferedReader for file io
      BufferedReader br = new BufferedReader(new FileReader(args[0]));

      //get the int constant for the file type
      int fileType = Integer.parseInt(args[1]);

      /*
       * get a Sequence Iterator over all the sequences in the file.
       * SeqIOTools.fileToBiojava() returns an Object. If the file read
```

```
       * is an alignment format like MSF and Alignment object is returned
       * otherwise a SequenceIterator is returned.
       */
      SequenceIterator iter =
          (SequenceIterator)SeqIOTools.fileToBiojava(fileType, br);

      //and now write it all to FASTA, (you can write to any OutputStream, not just
System.out)
      SeqIOTools.writeFasta(System.out, iter);
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

(Using Biojava 1.3)

```
import java.io.*;


import org.biojava.bio.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;

public class GeneralReader {

   /**
   * This program will read any file supported by SeqIOTools it takes three
   * arguments, the first is the file name the second is the format type the
   * third is the type of residue being read. Illegal combinations such as
   * SwissProt and DNA will cause an exception.
     *
   * Allowed formats are: (case insensitive).
     *
   * FASTA
   * EMBL
   * GENBANK
   * SWISSPROT (or swiss)
   * GENPEPT
     *
   * Allowed sequence types are: (case insensititve).
     *
   * DNA
   * AA (or Protein)
   * RNA
     *
     */
   public static void main(String[] args) {
       try {
           //prepare a BufferedReader for file io
      BufferedReader br = new BufferedReader(new FileReader(args[0]));
```

```
            //the flat file format
        String format = args[1];

            //the Alphabet
        String alpha = args[2];

            //get the int value for the format and alphabet


            /*
         * get a Sequence Iterator over all the sequences in the file.
         * SeqIOTools.fileToBiojava() returns an Object. If the file read
         * is an alignment format like MSF and Alignment object is returned
         * otherwise a SequenceIterator is returned.
               */
        SequenceIterator iter =
            (SequenceIterator)SeqIOTools.fileToBiojava(format, alpha, br);

            // do something with the sequences
        SeqIOTools.writeFasta(System.out, iter);
         }
         catch (FileNotFoundException ex) {
            //can't find file specified by args[0]
            ex.printStackTrace();
         }catch (BioException ex) {
            //invalid file format name
            ex.printStackTrace();
         }catch (IOException ex){
            //error writing to fasta
            ex.printStackTrace();
         }
    }
}
```

How can I turn an ABI trace into a BioJava Sequence?

A lot of Bioinformatics begins with the reading of a piece of DNA (or several pieces) using a DNA sequencer. A typical output is an ABI trace. BioJava contains a Class called ABITrace that will parse either an ABITrace file or URL or a byte[] and store its values for programmatic retrieval.

The following program is modified from a version kindly supplied by Matthew Pocock. It demonstrates the creation of a BioJava Sequence from an ABI trace file.

```
import java.io.*;

import org.biojava.bio.*;
import org.biojava.bio.program.abi.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.impl.*;
import org.biojava.bio.seq.io.*;
import org.biojava.bio.symbol.*;

public class Trace2Seq {
  public static void main(String[] args)
  throws Exception {
    File traceFile = new File(args[0]);

    //the name of the sequence
    String name = traceFile.getName();

    //read the trace
    ABITrace trace = new ABITrace(traceFile);

    //extract the Symbols
    SymbolList symbols = trace.getSequence();
    //make a fully fledged sequence
    Sequence seq = new SimpleSequence(symbols, name, name,
Annotation.EMPTY_ANNOTATION);

    //write it to STDOUT
    SeqIOTools.writeFasta(System.out, seq);
```

```
    }
}
```

How do I List the Annotations in a Sequence?

When you read in a annotates sequence file such as GenBank or EMBL there is a lot more detailed information in there than just the raw sequence. If the information has a sensible location then it ends up as a Feature. If it is more generic such as the species name then the information ends up as Annotations.

BioJava Annotation objects are a bit like Map objects and they contian key value mappings.

Below is the initial portion of an EMBL file

```
ID   AY130859   standard; DNA; HUM; 44226 BP.
XX
AC   AY130859;
XX
SV   AY130859.1
XX
DT   25-JUL-2002 (Rel. 72, Created)
DT   25-JUL-2002 (Rel. 72, Last updated, Version 1)
XX
DE   Homo sapiens cyclin-dependent kinase 7 (CDK7) gene, complete cds.
XX
KW   .
XX
OS   Homo sapiens (human)
OC   Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia;
OC   Eutheria; Primates; Catarrhini; Hominidae; Homo.
XX
RN   [1]
RP   1-44226
RA   Rieder M.J., Livingston R.J., Braun A.C., Montoya M.A., Chung M.-W.,
RA   Miyamoto K.E., Nguyen C.P., Nguyen D.A., Poel C.L., Robertson P.D.,
RA   Schackwitz W.S., Sherwood J.K., Witrak L.A., Nickerson D.A.;
RT   ;
RL   Submitted (11-JUL-2002) to the EMBL/GenBank/DDBJ databases.
RL   Genome Sciences, University of Washington, 1705 NE Pacific, Seattle, WA
RL   98195, USA
XX
CC   To cite this work please use:  NIEHS-SNPs, Environmental Genome
CC   Project, NIEHS ES15478, Department of Genome Sciences, Seattle, WA
CC   (URL: http://egp.gs.washington.edu).
```

The following program reads an EMBL file and lists its Annotation properties. The output of this program on the above file is listed below the program.

```
import java.io.*;
import java.util.*;
```

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

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

    try {
      //read in an EMBL Record
      BufferedReader br = new  BufferedReader(new FileReader(args[0]));
      SequenceIterator seqs = SeqIOTools.readEmbl(br);

      //for each sequence list the annotations
      while(seqs.hasNext()){
        Annotation anno = seqs.nextSequence().getAnnotation();

        //print each key value pair
        for (Iterator i = anno.keys().iterator(); i.hasNext(); ) {
          Object key = i.next();
          System.out.println(key +" : "+ anno.getProperty(key));
        }
      }
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

Program Output

```
RN : [1]
KW : .
RL : [Submitted (11-JUL-2002) to the EMBL/GenBank/DDBJ databases., Genome
Sciences, University of Washington, 1705 NE Pacific, Seattle, WA, 98195, USA]
embl_accessions : [AY130859]
DE : Homo sapiens cyclin-dependent kinase 7 (CDK7) gene, complete cds.
SV : AY130859.1
AC : AY130859;
FH : Key Location/Qualifiers
XX :
OC : [Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia;,
Eutheria; Primates; Catarrhini; Hominidae; Homo.]
RA : [Rieder M.J., Livingston R.J., Braun A.C., Montoya M.A., Chung M.-W.,,
Miyamoto K.E., Nguyen C.P., Nguyen D.A., Poel C.L., Robertson P.D.,, Schackwitz
W.S., Sherwood J.K., Witrak L.A., Nickerson D.A.;]
ID : AY130859 standard; DNA; HUM; 44226 BP.
DT : [25-JUL-2002 (Rel. 72, Created), 25-JUL-2002 (Rel. 72, Last updated, Version
```

1)]
CC : [To cite this work please use: NIEHS-SNPs, Environmental Genome, Project, NIEHS ES15478, Department of Genome Sciences, Seattle, WA, (URL: http://egp.gs.washington.edu).]
RT : ;
OS : Homo sapiens (human)
RP : 1-44226

How do I filter sequences based on their species?

The species field of a GenBank SwissProt or EMBL file ends up as an Annotation entry. Essentially all you need to do is get the species property from a sequences Annotation and check to see if it is what you want.

The species property name depends on the source: for EMBL or SwissProt it is "OS" for GenBank it is "Organism".

The following program will read in Sequences from a file and filter them according to their species. The same general recipe with a little modification could be used for any Annotation property.

```
import java.io.*;

import org.biojava.bio.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.db.*;
import org.biojava.bio.seq.io.*;

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

    try {
      //read an EMBL file specified in args[0]
      BufferedReader br = new BufferedReader(new FileReader(args[0]));
      SequenceIterator iter = SeqIOTools.readEmbl(br);

      //the species name to search for (specified by args[1]);
      String species = args[1];

      //A sequenceDB to store the filtered Seqs
      SequenceDB db = new HashSequenceDB();

      //As each sequence is read
      while(iter.hasNext()){
        Sequence seq = iter.nextSequence();
        Annotation anno = seq.getAnnotation();

        //check the annotation for Embl organism field "OS"
        if(anno.containsProperty("OS")){

          String property = (String)anno.getProperty("OS");

          //check the value of the property, could also do this with a regular
expression
          if(property.startsWith(species)){
```

```
            db.addSequence(seq);
          }
        }
      }

      //write the sequences as FASTA
      SeqIOTools.writeFasta(System.out, db);
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

How do I specify a PointLocation?

In BioJava locations in a Sequence are specified by simple objects that implement the interface Location.

A point location is the inclusive location of a single symbol in a SymbolList or Sequence. PointLocations have public constructors and are easy to instantiate. The following example demonstrates the creation of a PointLocation and it's specification of a single Symbol in a SymbolList.

Remember that BioJava uses the biological coordinate system thus the first PointLocation in a Sequence will be 1 not 0.

```
import org.biojava.bio.symbol.*;
import org.biojava.bio.seq.*;

public class SpecifyPoint {
  public static void main(String[] args) {
    try {
      //make a PointLocation specifying the third residue
      PointLocation point = new PointLocation(3);
      //print the location
      System.out.println("Location: "+point.toString());

      //make a SymbolList
      SymbolList sl = RNATools.createRNA("gcagcuaggcggaaggagc");
      System.out.println("SymbolList: "+sl.seqString());

      //get the SymbolList specified by the Location
      SymbolList sym = point.symbols(sl);
      //in this case the SymbolList will only have one base
      System.out.println("Symbol specified by Location: "+sym.seqString());
    }
    catch (IllegalSymbolException ex) {
      //illegal symbol used to make sl
      ex.printStackTrace();
    }
  }
}
```

How do I specify a RangeLocation?

In BioJava a RangeLocation is an object that holds the minimum and maximum bounds of a region on a SymbolList or Sequence. The minimum and maximum are inclusive.

The following example demonstrates the use of a RangeLocation.

```
import org.biojava.bio.symbol.*;
import org.biojava.bio.seq.*;

public class SpecifyRange {
  public static void main(String[] args) {
    try {
      //make a RangeLocation specifying the residues 3-8
      Location loc = LocationTools.makeLocation(3,8);
      //print the location
      System.out.println("Location: "+loc.toString());

      //make a SymbolList
      SymbolList sl = RNATools.createRNA("gcagcuaggcggaaggagc");
      System.out.println("SymbolList: "+sl.seqString());

      //get the SymbolList specified by the Location
      SymbolList sym = loc.symbols(sl);
      System.out.println("Symbols specified by Location: "+sym.seqString());
    }
    catch (IllegalSymbolException ex) {
      //illegal symbol used to make sl
      ex.printStackTrace();
    }
  }
}
```

How do CircularLocations work?

A number of interesting DNA molecules, such as plasmids and bacterial chromosomes are circular. Locations on a circular molecule are specified relative to some arbitrary origin.

In BioJava circular SymbolLists don't really exist. The underlying Symbols are ultimately stored as an array of pointers to Symbols. The circular effect can be faked using a CircularView object (which implements SymbolListView).

In a standard SymbolList it is not possible to access a Symbol using a Location that lies outside the SymbolList. Trying to get the Symbol at 0 or length+1 will throw an IndexOutOfBounds exception. In the case of a CircularView it is perfectly sensible to ask for the Symbol at 0 or -5 and expect to get a Symbol. Because BioJava uses the biological numbering system a Sequence is number from 0 to length.

No limits are placed on indexing a CircularView and a special convention is used for numbering. The Symbol indexed by 1 is the first Symbol in the underlying SymbolList. The Symbol indexed by 0 is the base immediately before the Symbol 1, which in this case is also the last base in the underlying SymbolList.

CircularLocations are dealt with using the CircularLocation class. CircularLocations are best constructed using the LocationTools class. This is demonstrated in the example below.

NOTE: due to bugs in earlier versions of BioJava this recipe will give strange results unless you are working off a fairly recent version of BioJava. To get the most up to date version follow the "How do I get and install BioJava" link on the main page and read the section on cvs. biojava-live BioJava version 1.3 (when released) will be adequate.

```
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

public class SpecifyCircular {
  public static void main(String[] args) {
    try {
      Location[] locs = new Location[3];
      //make a CircularLocation specifying the residues 3-8 on a 20mer
      locs[0] = LocationTools.makeCircularLocation(3,8,20);
      //make a CircularLocation specifying the residues 0-4 on a 20mer
      locs[1] = LocationTools.makeCircularLocation(0,4,20);
      //make a CircularLocation specifying the residues 18-24 on a 20mer
      locs[2] = LocationTools.makeCircularLocation(18,24,20);

      for (int i = 0; i &LT; locs.length; i++){
        //print the location
        System.out.println("Location: "+locs[i].toString());

        //make a SymbolList
        SymbolList sl = DNATools.createDNA("gcagctaggcggaaggagct");
```

```
        System.out.println("SymbolList: "+sl.seqString());

        //get the SymbolList specified by the Location
        SymbolList sym = locs[i].symbols(sl);
        System.out.println("Symbol specified by Location: "+sym.seqString());
      }
    }
    catch (IllegalSymbolException ex) {
      //illegal symbol used to make sl
      ex.printStackTrace();
    }
  }
}
```

How can I make a Feature?

In BioJava Features are a bit like an Annotation with a Location. There are various types of Features that all implement the Feature interface. All Feature implementations contain an inner class called 'Template'. The Template class specifies the minimum information needed to create a Feature. A feature is realized when the feature template is passed as an argument to the createFeature method of an implementation of the FeatureHolder interface.

Conveniently Sequence is a sub interface of FeatureHolder so it can hold features. Note that a SymbolList cannot hold Features. Interestingly the Feature interface is also a sub interface of FeatureHolder. Because of this a Feature can hold sub features in a nested hierarchy. This allows a 'gene' feature to hold 'exon' features and 'exon' features to hold 'snp' features etc. There is a built in safety check that will prevent a feature holding itself.

Feature templates can be created *de novo* or copied from an existing Feature. The following example shows both options.

```
import org.biojava.bio.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;
import org.biojava.utils.*;

public class MakeAFeature {
  public static void main(String[] args) {
    //get the feature template for a StrandedFeature
    StrandedFeature.Template templ = new StrandedFeature.Template();

    //fill in the template
    templ.annotation = Annotation.EMPTY_ANNOTATION;
    templ.location = new RangeLocation(3,6);
    templ.source = "my feature";
    templ.strand = StrandedFeature.POSITIVE;
    templ.type = "interesting motif";

    try {
      //the sequence the feature will go on
      Sequence seq = DNATools.createDNASequence("atgcgcttaag","seq1");
      System.out.println(seq.getName()+" contains "+seq.countFeatures()+" features");


      System.out.println("adding new feature...");

      //realize the feature on the Sequence and get a pointer to it so we can make
another
      Feature f = seq.createFeature(templ);
      System.out.println(seq.getName()+" contains "+seq.countFeatures()+" features");

      //make an identical template to that used to make f
      templ = (StrandedFeature.Template)f.makeTemplate();
      //give it a different location and type
      templ.location = new PointLocation(4);
      templ.type = "point mutation";


      System.out.println("adding nested feature...");
```

```
      //realize the new feature as a nested feature of f
      f.createFeature(templ);

      //notice how the countFeatures() method only counts top level features
      System.out.println(seq.getName()+" contains "+seq.countFeatures()+" features");
      System.out.println(f.getSource()+" contains "+seq.countFeatures()+" features");
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

How can I filter Features by type?

If you have just parsed a detailed Genbank file you will end up with a Sequence that contains several Features of different types. It may be that you are only interested in Features of the type "CDS" for example. To filter the Features you would use a FeatureFilter which can be used to generate a FeatureHolder containing only the Features that get past the FeatureFilter.

The following example shows the use of a "byType" FeatureFilter.

```
import java.util.*;

import org.biojava.bio.seq.*;

public class FilterByType {
  public static void main(String[] args) {
    Sequence seq = null;

  /*
   * code here to intitailize seq with numerous different features
   * possibly by reading a Genbank or similar file.
   *
   */

    //make a Filter for "CDS" types
    FeatureFilter ff = new FeatureFilter.ByType("CDS");

    //get the filtered Features
    FeatureHolder fh = seq.filter(ff);

    //iterate over the Features in fh
    for (Iterator i = fh.features(); i.hasNext(); ) {
      Feature f = (Feature)i.next();
    }
  }
}
```

How Do I Parse A BLAST Result?

Much of the credit for this example belongs to Keith James.

A frequent task in bioinformatics is the generation of BLAST search results. BioJava has the ability to parse "Blast-like" output such as Blast and HMMER using a trick that makes the BLAST output into SAX events that can be listened for by registered listeners.

The basic pipeline is as follows:

Blast_output -> Generate SAX events --> Convert SAX events --> Build result objects --> Store them in a list.

InputStream--> BLASTLikeSAXParser --> SeqSimilartyAdapter --> BlastLikeSearchBuilder --> List

The API is very flexible however for most purposes the following simple recipe will get you what you want.

```
import java.io.*;
import java.util.*;


import org.biojava.bio.program.sax.*;
import org.biojava.bio.program.ssbind.*;
import org.biojava.bio.search.*;
import org.biojava.bio.seq.db.*;
import org.xml.sax.*;
import org.biojava.bio.*;

public class BlastParser {
  /**
   * args[0] is assumed to be the name of a Blast output file
   */
  public static void main(String[] args) {
    try {
      //get the Blast input as a Stream
      InputStream is = new FileInputStream(args[0]);

      //make a BlastLikeSAXParser
      BlastLikeSAXParser parser = new BlastLikeSAXParser();

      //make the SAX event adapter that will pass events to a Handler.
      SeqSimilarityAdapter adapter = new SeqSimilarityAdapter();

      //set the parsers SAX event adapter
      parser.setContentHandler(adapter);

      //The list to hold the SeqSimilaritySearchResults
      List results = new ArrayList();

      //create the SearchContentHandler that will build SeqSimilaritySearchResults
      //in the results List
      SearchContentHandler builder = new BlastLikeSearchBuilder(results,
```

```
        new DummySequenceDB("queries"), new DummySequenceDBInstallation());

    //register builder with adapter
    adapter.setSearchContentHandler(builder);

    //parse the file, after this the result List will be populated with
    //SeqSimilaritySearchResults

    parser.parse(new InputSource(is));
    formatResults(results);
  }
  catch (SAXException ex) {
    //XML problem
    ex.printStackTrace();
  }catch (IOException ex) {
    //IO problem, possibly file not found
    ex.printStackTrace();
  }
}
```

How Do I Parse a FASTA Search Result?

The procedure for parsing FASTA results is very similar to the procedure for parsing BLAST results. If you take the recipe for the BLAST parser and replace this line

```
XMLReader parser = new BlastLikeSAXParser();
```

with

```
XMLReader parser = new FastaSearchSAXParser();
```

You will have a functional FASTA parser.

How Do I Extract Information From Search Results?

The Blast parsing and Fasta parsing procedures already discussed once the file is parsed a List of SeqSimilaritySearchResult objects. One of these is made per query. Each SeqSimilaritySearchResult contains a List of SeqSimilaritySearchHit objects which detail the hit from the Query to the Subject. Each SeqSimilaritySearchHit object contains a List of SeqSimilaritySearchSubHit objects. These are equivalent to the HSPs reported by BLAST.

The result, hit and subhits contain useful getXXX() methods to retrieve the stored information.

The code snippet below shows a private method that would take a List produced by a BLAST or FASTA parser and then extracts the hit id (subject id), its bit score and its e score.

```
  private static void formatResults(List results){

    //iterate through each SeqSimilaritySearchResult
    for (Iterator i = results.iterator(); i.hasNext(); ) {
      SeqSimilaritySearchResult result = (SeqSimilaritySearchResult)i.next();

      //iterate through the hits
      for (Iterator i2 = result.getHits().iterator(); i2.hasNext(); ) {
        SeqSimilaritySearchHit hit = (SeqSimilaritySearchHit)i2.next();

        //output the hit ID, bit score and e score
        System.out.println("subject:\t"+hit.getSubjectID() +
                           " bits:\t"+hit.getScore()+
                           " e:\t"+hit.getEValue());
      }
    }
  }
```

How Do I Count the Residues in a Sequence?

Counting the residues in a Sequence is a fairly standard bioinformatics task. Generally you would construct an array of ints and use some arbitrary indexing system. Better yet you could use an AlphabetIndex to impose a standardized index. You would get one from the AlphabetManager using one of its getAlphabetIndex() methods. Because this type of activity is so standard BioJava conveniently wraps up all the indexing etc into a class called IndexedCount which is an implementation of the Count interface.

The following program reads some type of sequence file and counts the residues, printing the results to STDOUT. Note that this program will not cope with ambiguity symbols. If you want to count ambiguity symbols you need add a partial count for each Symbol that makes up the ambiguity If this is the case you would use this solution.

Solution 1

```
import java.io.*;
import java.util.*;


import org.biojava.bio.dist.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;
import org.biojava.bio.symbol.*;

public class CountResidues {

  /**
   * Takes 2 arguments, first is a sequence filename the second is an int argument
   * that is equal to one of the formats supported by SeqIOTools. Suitable file types
   * are:
   * FASTADNA = 1;
   * FASTAPROTEIN = 2;
   * EMBL = 3;
   * GENBANK = 4;
   * SWISSPROT = 5;
   * GENPEPT = 6;
   */
  public static void main(String[] args) {
    //reference to object to hold the counts
    Count counts = null;

    try {
      //open sequence file
      BufferedReader br = new BufferedReader(new FileReader(args[0]));

      //get a SequenceIterator for the sequences in the file
      SequenceIterator iter = (SequenceIterator)SeqIOTools.fileToBiojava(
          Integer.parseInt(args[1]), br);

      //for each sequence
      while(iter.hasNext()){
```

```
        Sequence seq = iter.nextSequence();

        //if needed initialize counts
        if(counts == null){
          counts = new IndexedCount((FiniteAlphabet)seq.getAlphabet());
        }

        //iterate through the Symbols in seq
        for (Iterator i = seq.iterator(); i.hasNext(); ) {
          AtomicSymbol sym = (AtomicSymbol)i.next();
          counts.increaseCount(sym,1.0);
        }
      }

      //now print the results
      for (Iterator i = ((FiniteAlphabet)counts.getAlphabet()).iterator();
           i.hasNext(); ) {
        AtomicSymbol sym = (AtomicSymbol)i.next();
        System.out.println(sym.getName()+" : "+counts.getCount(sym));
      }
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

Solution 2

```
import java.io.*;
import java.util.*;


import org.biojava.bio.dist.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;
import org.biojava.bio.symbol.*;

public class CountResidues2 {

  /**
   * Takes 2 arguments, first is a sequence filename the second is an int argument
   * that is equal to one of the formats supported by SeqIOTools. Suitable file types
   * are:
   * FASTADNA = 1;
   * FASTAPROTEIN = 2;
   * EMBL = 3;
   * GENBANK = 4;
   * SWISSPROT = 5;
   * GENPEPT = 6;
   */
  public static void main(String[] args) {
    //reference to object to hold the counts
    Count counts = null;
```

```
    try {
      //open sequence file
      BufferedReader br = new BufferedReader(new FileReader(args[0]));

      //get a SequenceIterator for the sequences in the file
      SequenceIterator iter = (SequenceIterator)SeqIOTools.fileToBiojava(
          Integer.parseInt(args[1]), br);

      //for each sequence
      while(iter.hasNext()){
        Sequence seq = iter.nextSequence();

        //if needed initialize counts
        if(counts == null){
          counts = new IndexedCount((FiniteAlphabet)seq.getAlphabet());
        }

        //iterate through the Symbols in seq
        for (Iterator i = seq.iterator(); i.hasNext(); ) {
          Symbol sym = (Symbol)i.next();

          /*
           * The Symbol may be ambiguous so add a partial count for each Symbol
           * that makes up the ambiguity Symbol. Eg the DNA ambiguity n is made
           * of an Alphabet of four Symbols so add 0.25 of a count to each.
           */
          FiniteAlphabet subSymbols = (FiniteAlphabet)sym.getMatches();
          for (Iterator i2 = subSymbols.iterator(); i2.hasNext(); ) {
            AtomicSymbol sym2 = (AtomicSymbol)i2.next();
            counts.increaseCount(sym2, 1.0 / (double)subSymbols.size());
          }
        }
      }

      //now print the results
      for (Iterator i = ((FiniteAlphabet)counts.getAlphabet()).iterator();
           i.hasNext(); ) {
        AtomicSymbol sym = (AtomicSymbol)i.next();
        System.out.println(sym.getName()+" : "+counts.getCount(sym));
      }
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

How do I calculate the frequency of a Symbol in a Sequence?

One of the most useful classes in BioJava is the Distribution. A Distribution is a map from a Symbol to a frequency. Distributions are trained with observed Symbols using a DistributionTrainerContext. A DistributionTrainerContext can train several registered Distributions and will handle any Symbol from any Alphabet. Ambiguous Symbols are divided amongst the AtomicSymbols that make up the ambiguous BasisSymbol.

The following program demonstrates the training of three Distributions with Sequences from three different Alphabets.

```
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;
import org.biojava.bio.dist.*;
import org.biojava.utils.*;
import java.util.*;

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

    try {
      //make a DNA SymbolList
      SymbolList dna = DNATools.createDNA("atcgctagcgtyagcntatsggca");

      //make a RNA SymbolList
      SymbolList rna = RNATools.createRNA("aucgcuaucccaggga");

      //make a protein SymbolList
      SymbolList protein = ProteinTools.createProtein("asrvgchvhilmkapqrt");

      SymbolList[] sla = {dna, rna, protein};

      //get a DistributionTrainerContext
      DistributionTrainerContext dtc = new SimpleDistributionTrainerContext();

      //make three Distributions
      Distribution dnaDist =
          DistributionFactory.DEFAULT.createDistribution(dna.getAlphabet());
      Distribution rnaDist =
          DistributionFactory.DEFAULT.createDistribution(rna.getAlphabet());
      Distribution proteinDist =
          DistributionFactory.DEFAULT.createDistribution(protein.getAlphabet());
```

```
      Distribution[] da = {dnaDist, rnaDist, proteinDist};

      //register the Distributions with the trainer
      dtc.registerDistribution(dnaDist);
      dtc.registerDistribution(rnaDist);
      dtc.registerDistribution(proteinDist);

      //for each Sequence
      for (int i = 0; i &LT; sla.length; i++) {
        //count each Symbol to the appropriate Distribution
        for(int j = 1; j &LT;= sla[i].length(); j++){
          dtc.addCount(da[i], sla[i].symbolAt(j), 1.0);
        }
      }

      //train the Distributions
      dtc.train();

      //print the weights of each Distribution
      for (int i = 0; i &LT; da.length; i++) {
        for (Iterator iter = ((FiniteAlphabet)da[i].getAlphabet()).iterator();
             iter.hasNext(); ) {

          Symbol sym = (Symbol)iter.next();
          System.out.println(sym.getName()+" : "+da[i].getWeight(sym));
        }
        System.out.println("\n");
      }

    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

How can I turn a Count into a Distribution?

A Count can be simply converted into a Distribution by using the static countToDistribution() method from the DistributionTools class.

```
import org.biojava.bio.dist.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

public class count2Dist {
  public static void main(String[] args) {
    FiniteAlphabet alpha = RNATools.getRNA();
    AlphabetIndex index = AlphabetManager.getAlphabetIndex(alpha);

    try {
      //make a Count
      Count c = new IndexedCount(alpha);
      c.increaseCount(RNATools.a(),35.0);
      c.increaseCount(RNATools.c(),44.0);
      c.increaseCount(RNATools.g(),68.0);
      c.increaseCount(RNATools.u(),34.0);

      System.out.println("COUNT");
      for (int i = 0; i &LT; alpha.size(); i++) {
        AtomicSymbol s = (AtomicSymbol)index.symbolForIndex(i);
        System.out.println(s.getName()+" : "+c.getCount(s));
      }

      //make it into a Distribution
      Distribution d = DistributionTools.countToDistribution(c);

      System.out.println("\nDISTRIBUTION");
      for (int i = 0; i &LT; alpha.size(); i++) {
        Symbol s = index.symbolForIndex(i);
        System.out.println(s.getName()+" : "+d.getWeight(s));
      }
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
```

How can I turn a Count into a Distribution?

```
}
```

How can I generate a random Sequence from a Distribution?

BioJava Distribution objects have a method for sampling Symbols. By successively sampling enough Symbols you can build up a random sequence. Because this is a common task a static method is provided in DistributionTools called generateSequence().

The following program generates a random Sequence using a uniform Distribution over the DNA Alphabet. The emitted sequence will differ each time although its composition should be close to 25% of each residue. Non uniform distributions can be used to generate biased sequences.

```
import org.biojava.bio.dist.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;
import java.io.*;

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

    //make a uniform distribution over the DNA Alphabet
    Distribution dist = new UniformDistribution(DNATools.getDNA());

    //generate a 700bp random sequence
    Sequence seq = DistributionTools.generateSequence("random seq", dist, 700);

    try {
      //print it to STDOUT
      SeqIOTools.writeFasta(System.out, seq);
    }
    catch (IOException ex) {
      //io error
      ex.printStackTrace();
    }
  }
}
```

How can I find the amount of information or entropy in a Distribution?

The amount of information or entropy in a Distribution is a reflection of the redundancy of the Distribution. Shannon information and Entropy can be calculated using static methods from the DistributionTools class.

Shannon information is returned as a double and reflects the total information content. The entropy is returned as a HashMap between each Symbol and its corresponding entropy. The following program calculates both for a very biased Distribution.

```
import java.util.*;

import org.biojava.bio.dist.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

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

    Distribution dist = null;
    try {
      //create a biased distribution
      dist =
          DistributionFactory.DEFAULT.createDistribution(DNATools.getDNA());

      //set the weight of a to 0.97
      dist.setWeight(DNATools.a(), 0.97);

      //set the others to 0.01
      dist.setWeight(DNATools.c(), 0.01);
      dist.setWeight(DNATools.g(), 0.01);
      dist.setWeight(DNATools.t(), 0.01);
    }
    catch (Exception ex) {
      ex.printStackTrace();
      System.exit(-1);
    }

    //calculate the information content
    double info = DistributionTools.bitsOfInformation(dist);
    System.out.println("information = "+info+" bits");
    System.out.print("\n");
```

```
    //calculate the Entropy (using the conventional log base of 2)
    HashMap entropy = DistributionTools.shannonEntropy(dist, 2.0);

    //print the Entropy of each residue
    System.out.println("Symbol\tEntropy");
    for (Iterator i = entropy.keySet().iterator(); i.hasNext(); ) {
      Symbol sym = (Symbol)i.next();
      System.out.println(sym.getName()+ "\t" +entropy.get(sym));
    }
  }
}
```

What is an easy way to tell if two Distributions have equal weights?

Testing two distributions for equal weights is a good way of telling if a training procedure has converged or if two Sequences are likely to come from the same organism. It is a bit tedious to loop through all the residues, especially in a large Alphabet. Fortunately there is a static method called areEmissionSpectraEqual() in DistributionTools that checks for you.

Using this method is demonstrated below.

```
import org.biojava.bio.dist.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;
import org.biojava.bio.*;
import org.biojava.utils.*;

public class EqualDistributions {
  public static void main(String[] args) {
    FiniteAlphabet alpha = DNATools.getDNA();

    //make a uniform distribution
    Distribution uniform = new UniformDistribution(alpha);

    try {
      //make another Distribution with uniform weights
      Distribution dist = DistributionFactory.DEFAULT.createDistribution(alpha);
      dist.setWeight(DNATools.a(), 0.25);
      dist.setWeight(DNATools.c(), 0.25);
      dist.setWeight(DNATools.g(), 0.25);
      dist.setWeight(DNATools.t(), 0.25);

      //test to see if the weights are equal
      boolean equal = DistributionTools.areEmissionSpectraEqual(uniform, dist);
      System.out.println("Are 'uniform' and 'dist' equal? "+ equal);
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

How do I make a custom Alphabet then take an OrderNDistribution over it?

This example demonstrates the creation of a custom Alphabet that will have seven Symbols. The custom made Symbols and Alphabet can then be used to make SymbolLists, Sequences, Distributions etc. When the AlphabetManager creates the CrossProductAlphabet, it will infer that the order of the conditioning alphabet is (order - 1) and the order of the conditioned alphabet is 1.

Contributed by Russell Smithies.

```
import java.io.*;
import java.util.*;

import org.biojava.bio.*;
import org.biojava.bio.dist.*;
import org.biojava.bio.symbol.*;
import org.biojava.utils.*;

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

    //create a custom dwarf Alphabet
    String[] dNames = {
        "Grumpy", "Sleepy", "Dopey", "Doc", "Happy", "Sneezy", "Bashful"
    };
    Symbol[] dwarfs = new Symbol[7];
    SimpleAlphabet dwarfAlphabet = new SimpleAlphabet();

    //give the new Alphabet a name
    dwarfAlphabet.setName("Dwarf");

    for (int i = 1; i &LT;= 7; i++) {
      try {
        dwarfs[i - 1] = AlphabetManager.createSymbol((char) ('0' + i), "" + dNames[i
- 1],Annotation.EMPTY_ANNOTATION);
         //add your new Symbols to the Alphabet
                        dwarfAlphabet.addSymbol(dwarfs[i - 1]);
      }
      catch (Exception e) {
        throw new NestedError(e, "Can't create symbols to represent dwarf");
      }

    //it is usual (but not essential) to register newly creates Alphabets with the
AlphabetManager
    AlphabetManager.registerAlphabet(dwarfAlphabet.getName(), dwarfAlphabet);
```

```
    }
```

Create an OrderNDstribution using the newly built Dwarf Alphabet

```
    //order of the distribution
    int order = 3;

    //create the cross-product Alphabet
    Alphabet a = AlphabetManager.getCrossProductAlphabet(Collections.nCopies(order,
dwarfAlphabet));

    //use the OrderNDistributionFactory to create the Distribution
    OrderNDistribution ond =
(OrderNDistribution)OrderNDistributionFactory.DEFAULT.createDistribution(a);

    //create the DistributionTrainer
    DistributionTrainerContext dtc = new SimpleDistributionTrainerContext();

    //register the Distribution with the trainer
    dtc.registerDistribution(ond);
```

This shows the creation of of a SymbolList from the Dwarf Alphabet so we can test our new OrderNDistribution. This is done by making, a UniformDistribution which is randomly sampled and adding the Symbols to an ArrayList. The ArrayList is then used to build the SymbolList.

```
    //create a random symbolList of dwarves
    UniformDistribution udist = new
UniformDistribution((FiniteAlphabet)dwarfAlphabet);

    int size = 100;
    List list = new ArrayList();

    for (int i = 0; i &LT; size; i++) {
      list.add(udist.sampleSymbol());
    }

    //create a symbolList to test the Distribution
    SymbolList symbl = new SimpleSymbolList(dwarfAlphabet, list);
```

The SymbolList is changed into an OrderNSymbolList to enable an OrderNDistribution to be made over it.

```
    //make it into an orderNSymbolList
    symbl = SymbolListViews.orderNSymbolList(symbl, order);

    //or you could have a windowed symbolList
    //symbl = SymbolListViews.windowedSymbolList(symbl, order);

    //add counts to the distribution
    for (Iterator i = symbl.iterator(); i.hasNext(); ) {
      try {
```

```
        dtc.addCount(ond, (Symbol) i.next(), 1.0);
      }
      catch (IllegalSymbolException ex) {
       //you  tried to add a Symbol not in your Alphabet
        ex.printstacktrace()}
    }

    // don't forget to train or none of your weights will be added
    dtc.train();

    //write the distribution to XML
    XMLDistributionWriter writer = new XMLDistributionWriter();

    writer.writeDistribution(ond, new FileOutputStream("dwarf.xml"));
  }
}
```

How can I write a Distribution to XML?

If you frequently construct Distributions from large training sets for later analysis it is desirable to be able to store these Distributions for latter use. One possibility is to serialize the Distribution to binary. Serialization, while ideal for short term storage or communication between Java VMs, is fragile and likely to break between different versions of BioJava. It is also impossible to inspect by eye.

A better solution is write the Distribution to XML, providing a long term, human readable and language independent solution. The following example shows how a Distribution can be written to XML and read back again. The example requires a fairly recent version of BioJava as the XMLDistributionWriter and XMLDistributionReader are fairly new features. The cvs version or version 1.3 (when released) will be adequate.

```
import java.io.*;

import org.biojava.bio.dist.*;
import org.biojava.bio.seq.*;

public class Dist2XMLandBack {
  public static void main(String[] args) {
    XMLDistributionWriter writer = new XMLDistributionWriter();
    XMLDistributionReader reader = new XMLDistributionReader();

    try {
      File temp = File.createTempFile("xmltemp",".xml");

      //create a Distribution to write
      Distribution d =
          DistributionFactory.DEFAULT.createDistribution(DNATools.getDNA());

      //give the Distribution some random values
      DistributionTools.randomizeDistribution(d);

      //write it to 'temp'
      writer.writeDistribution(d, new FileOutputStream(temp));

      //read it back in
      Distribution d2 = reader.parseXML(new FileInputStream(temp));

      //check that the weights are reproduced
      boolean b = DistributionTools.areEmissionSpectraEqual(d,d2);
      System.out.println("Are values reproduced? "+b);
    }
    catch (Exception ex) {
      ex.printStackTrace();
```

```
      }
    }
}
```

How do I use a WeightMatrix to find a motif?

A Weight Matrix is a useful way of representing an alignment or a motif. It can also be used as a scoring matrix to detect a similar motif in a sequence. BioJava contains a class call WeightMatrix in the org.biojava.bio.dp package. There is also a WeightMatrixAnnotator which uses the WeightMatrix to add Features to any portion of the sequence being searched which exceed the scoring threshold.

The following program generates a WeightMatrix from an aligment and uses that matrix to annotate a Sequence with a threshold of 0.1

```
import java.util.*;


import org.biojava.bio.dist.*;
import org.biojava.bio.dp.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

public class WeightMatrixDemo {
  public static void main(String[] args) throws Exception{
    //make an Alignment of a motif.
    Map map = new HashMap();
    map.put("seq0", DNATools.createDNA("aggag"));
    map.put("seq1", DNATools.createDNA("aggaa"));
    map.put("seq2", DNATools.createDNA("aggag"));
    map.put("seq3", DNATools.createDNA("aagag"));
    Alignment align = new SimpleAlignment(map);

    //make a Distribution[] of the motif
    Distribution[] dists =
        DistributionTools.distOverAlignment(align, false, 0.01);

    //make a Weight Matrix
    WeightMatrix matrix = new SimpleWeightMatrix(dists);

    //the sequence to score against
    Sequence seq = DNATools.createDNASequence("aaagcctaggaagaggagctgat","seq");

    //annotate the sequence with the weight matrix using a low threshold (0.1)
    WeightMatrixAnnotator wma = new WeightMatrixAnnotator(matrix, 0.1);
    seq = wma.annotate(seq);

    //output match information
    for (Iterator it = seq.features(); it.hasNext(); ) {
      Feature f = (Feature)it.next();
      Location loc = f.getLocation();
      System.out.println("Match at " + loc.getMin()+"-"+loc.getMax());
```

```
      System.out.println("\tscore : "+f.getAnnotation().getProperty("score"));
    }
  }
}
```

How do I make a ProfileHMM?

Profile HMMs (such as those used in the program HMMER) are very sensitive tools for searching for motifs. A profile HMM is typically trained from a set of input sequences that contain the motif of interest using the Baum-Welch algorithm. This algorithm optimises the parameters of the model until some stopping criteria is satisfied. Once a profile HMM has been constructed the Viterbi algorithm can be used to determine the state path most likely to have generated an observed (test) sequence. If sufficient match states are observed the test sequence can be deemed to contain the motif, alternatively some scoring metric can be used (such as log odds) and a cutoff threshold defined. The following demonstrates the construction and use of a ProfileHMM in BioJava.

The first step is to create the profile HMM.

```
    /*
     * Make a profile HMM over the DNA Alphabet with 12 'columns' and default
     * DistributionFactories to construct the transition and emmission
     * Distributions
     */
    ProfileHMM hmm = new ProfileHMM(DNATools.getDNA(),
                         12,
                         DistributionFactory.DEFAULT,
                         DistributionFactory.DEFAULT,
                         "my profilehmm");

    //create the Dynamic Programming matrix for the model.
    dp = DPFactory.DEFAULT.createDP(hmm);
```

At this point you would read in a set of sequences that make up the training set.

```
    //Database to hold the training set
    SequenceDB db = new HashSequenceDB();

    //code here to load the training set
```

Now initialize all of the model parameters to a uniform value. Alternatively parameters could be set randomly or set to represent a guess at what the best model might be. Then use the Baum-Welch Algorithm to optimise the parameters.

```
    //train the model to have uniform parameters
    ModelTrainer mt = new SimpleModelTrainer();
    //register the model to train
    mt.registerModel(hmm);
    //as no other counts are being used the null weight will cause everything to be
uniform
    mt.setNullModelWeight(1.0);
    mt.train();

    //create a BW trainer for the dp matrix generated from the HMM
    BaumWelchTrainer bwt = new BaumWelchTrainer(dp);

    //anonymous implementation of the stopping criteria interface to stop after 20
iterations
```

```
StoppingCriteria stopper = new StoppingCriteria(){
  public boolean isTrainingComplete(TrainingAlgorithm ta){
    return (ta.getCycle() &GT; 20);
  }
};

/*
 * optimize the dp matrix to reflect the training set in db using a null model
 * weight of 1.0 and the Stopping criteria defined above.
 */
bwt.train(db,1.0,stopper);
```

Below is an example of scoring a sequence and outputting the state path.

```
SymbolList test = null;
//code here to initialize the test sequence

/*
 * put the test sequence in an array, an array is used because for pairwise
 * alignments using an HMM there would need to be two SymbolLists in the
 * array
 */

SymbolList[] sla = {test};

//decode the most likely state path and produce an 'odds' score
StatePath path = dp.viterbi(sla, ScoreType.ODDS);
System.out.println("Log Odds = "+path.getScore());

//print state path
for(int i = 1; i &LT;= path.length(); i++){
  System.out.println(path.symbolAt(StatePath.STATES, i).getName());
}
```

How can I view the Features and Annotations as a tree?

Given that Sequences can hold Annotations, with their key value pairs, and Features, and that Features can hold information, Annotations and nested Features, which can contain still more annotations, nested features etc it would be useful to be able to view it all as a structured tree.

Fortunately the friendly BioJava team have made the FeatureTree class to let you see where all that structure goes. The FeatureTree extends the JTree component and can easily be used in a GUI. The data used by the tree is supplied in the form of a SequenceDB that can be made by reading a text file.

The following program demonstrates the use of a FeatureTree. It takes two arguments. The first is the name of a file containing sequence data. The second is a number specifying the format of the data.

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;

import javax.swing.*;

import org.biojava.bio.gui.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.db.*;
import org.biojava.bio.seq.io.*;

public class TreeFrame extends JFrame {
  private JPanel jPanel = new JPanel();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private BorderLayout borderLayout = new BorderLayout();
  private FeatureTree featureTree = new FeatureTree();

  public TreeFrame() {
    try {
      init();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * This program will read files supported by SeqIOTools and display its
   * Sequence, Annotations and Features as a Tree. It takes two
   * arguments, the first is the file name the second is the int constant
   * for the file type in SeqIOTools. See SeqIOTools for possible file types.
   * Valid constants for this program are:
   *
   * FASTADNA = 1;
   * FASTAPROTEIN = 2;
   * EMBL = 3;
```

```
   * GENBANK = 4;
   * SWISSPROT = 5;
   * GENPEPT = 6;
   *
   */
  public static void main(String[] args) throws Exception{

    //read the sequence flat file
    BufferedReader br = new BufferedReader(new FileReader(args[0]));
    //get the format type from the command line
    int type = Integer.parseInt(args[1]);

    //read the sequences into a DB that will serve as the model for the tree
    SequenceDB db = new HashSequenceDB();
    SequenceIterator iter = (SequenceIterator)SeqIOTools.fileToBiojava(type, br);
    while(iter.hasNext()){
      db.addSequence(iter.nextSequence());
    }
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    TreeFrame treeFrame = new TreeFrame();
    //set the SequenceDB to serve as the data model
    treeFrame.getFeatureTree().setSequenceDB(db);
    treeFrame.pack();
    treeFrame.show();
  }

  private void init() throws Exception {
    jPanel.setLayout(borderLayout);
    this.setTitle("FeatureTree Demo");
    this.getContentPane().add(jPanel, BorderLayout.CENTER);
    jPanel.add(jScrollPane1,  BorderLayout.CENTER);
    jScrollPane1.getViewport().add(featureTree, null);
  }

  public FeatureTree getFeatureTree() {
    return featureTree;
  }

  protected void processWindowEvent(WindowEvent we){
    if(we.getID() == WindowEvent.WINDOW_CLOSING){
      System.exit(0);
    }else{
      super.processWindowEvent(we);
    }
  }
}
```

How can I display a Sequence in a GUI

When building a bioinformatics GUI you will probably want to display the sequence of residues in the Sequence you are displaying. BioJava contains a number of GUI components that can render various aspects of a Sequence.

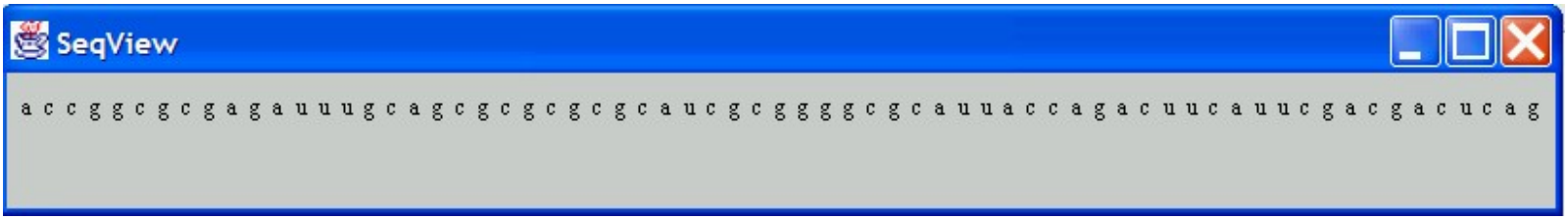
The basic unit of any Sequence based GUI is the SequenceRenderContext which holds the Sequence and sends instructions to a SequenceRenderer which does the actual drawing of the Sequence. There are several SequenceRenderer implementations in BioJava. The one to display the order of residues is the SymbolSequenceRenderer.

The following program demonstrates the use of a SequenceRenderContext and a SequenceRenderer to display the symbols in a Sequence.

Below the program is a screen shot of the GUI.

```
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import org.biojava.bio.gui.sequence.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

public class SeqView extends JFrame {
  private Sequence seq;
  private JPanel jPanel = new JPanel();
  private SequencePanel seqPanel = new SequencePanel();
  private SequenceRenderer symSeqRenderer = new SymbolSequenceRenderer();

  public SeqView() {
    try {
      //create the sequence to display
      seq = RNATools.createRNASequence("accggcgcgagauuugcagcgcgcgcgcaucgcg"+
                                       "gggcgcauuaccagacuucauucgacgacucagc"
                                       ,"rna1");
      init();
    }
    catch(Exception e) {
      e.printStackTrace();
    }

  }
  public static void main(String[] args) {
    SeqView seqView = new SeqView();
    seqView.pack();
    seqView.show();
  }

  /**
   * Set up the components to display the graphics
   */
  private void init() throws Exception {
    this.getContentPane().setLayout(new BorderLayout());
    this.getContentPane().add(jPanel, BorderLayout.CENTER);
    this.setTitle("SeqView");
    jPanel.add(seqPanel, BorderLayout.CENTER);

    //set the sequence to display
    seqPanel.setSequence(seq);
```

```
    //set the object responsible for painting the sequence
    seqPanel.setRenderer(symSeqRenderer);

    //the amount of sequence to display
    seqPanel.setRange(new RangeLocation(1,seq.length()));
  }

  /**
   * Overide this to close the program when the window closes.
   */
  protected void processWindowEvent(WindowEvent we){
    if (we.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
    else {
      super.processWindowEvent(we);
    }
  }
}
```



How do I display Sequence coordinates?

When displaying a sequence it is useful to display the coordinates of the sequence so you can tell where you are up to. BioJava contains a SequenceRenderer implementation called a RulerRenderer that displays Sequence coordinates.

Because a SequenceRenderContext can only use a single SequenceRenderer at a time you will need to use a MultiLineRenderer. A MultiLineRenderer implements SequenceRenderer and can wrap up multiple SequenceRenderers coordinating their displays as several tracks.

The use of a RulerRenderer and a MultiLineRenderer is demonstrated in the program below. A screen shot of the GUI is displayed below the program.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import org.biojava.bio.gui.sequence.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

public class MultiView extends JFrame {
  private JPanel jPanel = new JPanel();
  private MultiLineRenderer mlr = new MultiLineRenderer();
  private SequenceRenderer symR = new SymbolSequenceRenderer();
  private RulerRenderer ruler = new RulerRenderer();
  private SequencePanel seqPanel = new SequencePanel();
  private Sequence seq;


  public MultiView() {
    try {
      seq = ProteinTools.createProteinSequence(
          "agcgstyravlivtymaragrsecharlvahklchg",
          "protein 1");
      init();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  public static void main(String[] args) {
    MultiView multiView = new MultiView();
    multiView.pack();
    multiView.show();
  }

  /**
```

```
   * OverRide to allow termination of program.
   */
  protected void processWindowEvent(WindowEvent we){
    if (we.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
    else {
      super.processWindowEvent(we);
    }
  }

  /**
   * Set up GUI components
   */
  private void init() throws Exception {
    this.setTitle("MultiView");
    this.getContentPane().add(jPanel, BorderLayout.CENTER);
    jPanel.add(seqPanel, BorderLayout.CENTER);

    //add the SymbolSequenceRenderer and RulerRenderer to the MultiLineRenderer
    mlr.addRenderer(symR);
    mlr.addRenderer(ruler);

    //set the MultiLineRenderer as the main renderer
    seqPanel.setRenderer(mlr);

    //set the Sequence
    seqPanel.setSequence(seq);

    //set the range to show
    seqPanel.setRange(new RangeLocation(1,seq.length()));
  }
}
```

How do I display Features?

Features are displayed by implementations of the FeatureRenderer interface. FeatureRenderers work in much the same way as SequenceRenderers and handle the drawing of the Features from a Sequence that is held in a SequenceRenderContext.

A SequenceRenderContext has no way of interacting directly with a FeatureRenderer so a FeatureBlockSequenceRenderer is used to wrap up the FeatureRenderer and act as a proxy.

The use of a FeatureBlockSequenceRenderer and a FeatureRenderer is demonstrated in the program below. A screen shot follows the program.

```
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import org.biojava.bio.*;
import org.biojava.bio.gui.sequence.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.symbol.*;

public class FeatureView extends JFrame {
  private Sequence seq;
  private JPanel jPanel1 = new JPanel();

  private MultiLineRenderer mlr = new MultiLineRenderer();
  private FeatureRenderer featr = new BasicFeatureRenderer();
  private SequenceRenderer seqR = new SymbolSequenceRenderer();
  private SequencePanel seqPanel = new SequencePanel();
  //the proxy between featr and seqPanel
  private FeatureBlockSequenceRenderer fbr = new FeatureBlockSequenceRenderer();

  public FeatureView() {
    try {
      seq = DNATools.createDNASequence(
          "atcgcgcatgcgcgcgcgcgcgctttatagcgatagagatata",
          "dna 1");

      //create feature from 10 to 25
      StrandedFeature.Template temp = new StrandedFeature.Template();
      temp.annotation = Annotation.EMPTY_ANNOTATION;
      temp.location = new RangeLocation(10,25);
      temp.source = "";
      temp.strand = StrandedFeature.POSITIVE;
      temp.type = "";

      //create another from 30 to 35
      Feature f = seq.createFeature(temp);
      temp = (StrandedFeature.Template)f.makeTemplate();
      temp.location = new RangeLocation(30,35);
```

```
      temp.strand = StrandedFeature.NEGATIVE;
      seq.createFeature(temp);

      //setup GUI
      init();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    FeatureView featureView = new FeatureView();
    featureView.pack();
    featureView.show();
  }

  /**
   * initialize GUI components
   */
  private void init() throws Exception {
    this.setTitle("FeatureView");
    this.getContentPane().add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(seqPanel, null);

    //Register the FeatureRenderer with the FeatureBlockSequenceRenderer
    fbr.setFeatureRenderer(featr);

    //add Renderers to the MultiLineRenderer
    mlr.addRenderer(fbr);
    mlr.addRenderer(seqR);

    //set the MultiLineRenderer as the SequencePanels renderer
    seqPanel.setRenderer(mlr);

    //set the Sequence to Render
    seqPanel.setSequence(seq);

    //display the whole Sequence
    seqPanel.setRange(new RangeLocation(1,seq.length()));
  }

  /**
   * Overridden so program terminates when window closes
   */
  protected void processWindowEvent(WindowEvent we){
    if (we.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
    else {
      super.processWindowEvent(we);
    }
```

How do I display features?

```
    }
}
```