DOCS AND HELP

http://www.biojava.org/ particularly the cookbook and the BioJavaX docs. The latter in particular will give you ideas as to how to design the new phylo code. API JavaDocs are also on the website.

CVS

We will work on the CVS head with the aim of including our code in the 1.6 release (the 1.5 release is currently in beta on a branch of its own). Please do not commit anything that does not compile.

```
CVSRSH=ssh
cvs -d :ext:username@dev.open-bio.org:/home/repository/biojava co biojava-live
```

(Username to be supplied – mine is 'holland').

CLASSPATH

Make sure you have all the JAR files in the ant-build folder included on your classpath when running BioJava apps. This folder won't get created until after you've build BioJava.

BUILDING

You'll need Ant, from http://jakarta.apache.org/ant/

Use Java 1.4. BioJava is not developed or tested using newer versions yet.

```
ant build
```

This creates the ant-build folder inside the root folder and puts all the dependency JAR files in there, as well as the newly compiled biojava.jar.

TESTING

JUnit tests are provided for existing code, but hopefully we will not be touching any existing code.

JUnit tests for new code developed at the hackathon are a good idea, and if we go down that path then we need to make sure we run them before every major commit.

To run tests you will need Junit 3.8.1 or earlier installed. JUnit 4.0.0 and later were compiled with Java 1.5 and will not work with the Java 1.4 binaries.

http://junit.sourceforce.net/

To run the tests, add the JUnit jar to your classpath then change to the biojava-live root directory and type:

```
ant runtests
```

SOURCE FORMATTING

We ask that you format code clearly. There is no particular convention it seems, as each developer that contributes uses slightly different styles. However, it must be formatted in a consistent and obvious manner and well commented.

CODE CONVENTIONS

Phyloinformatics-specific code will go in the org.biojavax.bio.phylo package. IO code will go in the org.biojavax.bio.phylo.io package. Custom exceptions go in org.biojavax.bio.exceptions. Code that could potentially be reused for other purposes may go in one of the existing org.biojavax packages.

These phylo packages doesn't exist yet – whoever gets there first will have the honour of creating them!

Everything must be defined by an interface, which is then implemented either in another class, or inside the interface in an example class. This is to make it easy in future to change our minds about implementation without having to redefine client code.

Fields are lower case unless they are acronyms, in which case use your discretion. Methods are lower case with inner capitals to denote new words. Use JavaBeans-style getters/setters wherever possible (getXYZ/isXYZ, setXYZ) inside the code. Make sure you make private everything that is specific to the internals of the class, and think carefully about the appropriate scope to use elsewhere.

Personally I am a fan of always prefixing references to local methods and fields with 'this.', or for static classes using the classname. It makes it a lot easier to work out where fields in particular are living and what scope the reference will affect. However this doesn't seem to be common amongst other developers on BioJava. Use your discretion!

Exceptions should inherit from org.biojava.bio.BioException/BioRuntimeException/BioError (note this is biojava not biojavax), or should be one of the existing exceptions found elsewhere in the biojava/biojavax packages. Always throw exceptions for errors, and never use the return value to indicate failure unless multiple statuses are possible.

If a method uses a parameter that can be one of a range of constants, define those constants as fields and refer to them everywhere via the field instead of by value. (e.g. instead of passing in 1,2,3, pass in constant fields with appropriate names, and validate the input by checking against the fields instead of actual values).

We make lots of use of the factory pattern, particularly in file parsing. We also make extensive use of the singleton pattern, using the private constructor plus static instance map method.

Toolkits that work generically on all implementations of a particular interface are implemented as static methods inside a Tools class in the inteface itself. This includes simple IO wrappers. See org.biojavax.bio.seq.RichSequence.

FILE PARSING

The existing biojavax file parsers are written using callback mechanisms, and have a single method available to client code that reads the next object from the file, or returns null if there is no next object.

The file parser knows how to recognise the start and end of an object's worth of data within the file. It calls methods on some object builder (e.g. org.biojavax.bio.seq.io.RichSequenceBuilder) to start/end an object, and as it reads the file it passes in the data that it has read to various methods in the object builder. The object builder uses this data to construct an object which will be returned when the end method is called. The file parser then returns this constructed object to the client code.

DATABASE INTERACTION

Currently, the only database BioJava will interact with is BioSQL, and this is achieved using Hibernate. The current code accesses the whole of the currently known schema and will not need modifying.

However, if we wish to access other databases for the purposes of the hackathon, we can copy the logic and ideas from the BioSQL code and create new packages in org.biojavax.bio.phylo.db. Hibernate 3.1 is the version that BioJava/BioSQL was developed and tested against.

SEQUENCE PARSING

BioJava has Alphabet and SymbolList interfaces which define sequence strings in a compact manner. This is much better than storing them as strings, as many methods are already defined for manipulating these efficiently. It also allows ambiguities to be decoded easily, and for identical symbols in different alphabets to be differentiated (e.g. 'A' in DNA and Protein). See Mark's presentation which is linked from the cookbook on the website.

If we need to use sequences in our code, then we need to use the org.biojavax.bio.seq implementations rather than the older ones from org.biojava.bio.seq. Likewise annotations and features, locations and anything else that has newer biojavax alternatives.

We use SAX to parse XML, and there are some tools for writing XML in org.biojava.utils.xml.

COMMAND LINE INTERACTION

BioJava has existing packages for this at org.biojava.utils.process. We may need to investigate if these need improving, or if they are already OK. Any new command line tool interface classes should go in org.biojavax.bio.phylo.program.

OTHER TOOLS

The original biojava packages include some distribution classes, and the newer biojavax packages have some cool genetic algorithm stuff. These might be useful if we have to do any tree construction work, although I don't know much about them myself – Mark is the expert here.

STAYING IN TOUCH

We should use the biojava-dev mailing list to discuss issues that may impact other people's code. You can sign up for this via the mailing lists page on http://www.biojava.org/ .

During the hackathon, you can contact me by email at: dicknetherlands@gmail.com