

**J-Features:
Development of a Java-based Feature
Table Editor**

A thesis submitted to the University of Manchester for the degree of
Master of Science in the School of Biological Sciences, Faculty of
Science

William S. J. Valdar, 1998

Contents

	Page
Abstract	3
Declaration	4
Copyright Notice	5
Lists of Figures and Tables	6
Acknowledgments	8
Introduction	9
Requirements for a feature table editor	10
Interface strategies	10
Aims	18
Implementation	19
Languages and platforms	19
Design of J-Features	22
J-Features: A user perspective	24
Discussion	35
I/O model	36
Graphical widgets	37
User interface	40
Extending J-Features	41
Conclusion	42
References	43
Appendix 1: Class Descriptions	46
Appendix 2: Class Hierarchy	48
Appendix 3: J-Features Source Code	52

Abstract

Feature tables are used by a number of biological data banks to hold information about the structural and functional properties of a sequence. At present, the formats of feature tables are not consistent, varying widely between different data banks. This makes it difficult for a scientist to add their knowledge to the data bank and suggests the need for a computational tool to make these inconsistencies transparent. An interactive feature table editor program was written in Java. The editor runs as an applet from a Web page and allows the user to add to, edit and delete the contents of a feature table. The program accepts input and saves output in Swiss-Prot format, but may be easily modified for other data bank formats. The editor includes interactive visualization tools that help navigate the information space of the sequence annotation. The program is most suitable as a prototype application that can be extended both in terms of its scope and its functionality.

Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Notice

(1) Copyright in text of this thesis rests with the Author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the Author and lodged in the John Rylands University Library of Manchester. Details may be obtained from the Librarian. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the Author.

(2) The ownership of any intellectual property rights which may be described in this thesis is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

Further information on the conditions under which disclosures and exploitation may take place is available from the Head of the School of Biological Sciences.

Lists of Figures and Tables

Figures:

1. Screen shot from ECGG Newfeatures. 12
2. Screen shot from the feature table editor in SeqLab. 14
3. Screen shot from the Feature Viewer applet in JaMBW. 16
4. Schematic diagram of the structure of the J-Features program. 23
5. Screen shot from the J-Features applet: Web page. 25
6. Screen shot from the J-Features applet: title frame. 26
7. Screen shot from the J-Features applet: input frame. 26
8. Screen shot from the J-Features applet: feature diagram. 27
9. Screen shot from the J-Features applet: sequence diagram. 27
10. Screen shots from the J-Features applet: selecting features on the feature diagram. 28
11. Screen shot from the J-Features applet: feature information frame. 29
12. Screen shot from the J-Features applet: a highlighted region in the sequence diagram. 29
13. Screen shot from the J-Features applet: dialog box requesting confirmation of deletion of a feature. 29
14. Screen shot from the J-Features applet: feature input form. 30
15. Screen shot from the J-Features applet: customize features frame. 31
16. Screen shot from the J-Features applet: customize sequence colours. 32
17. Screen shot from the J-Features applet: output frame. 33

Tables:

Appendix 1: Class Descriptions

Acknowledgments

The work presented in this report was produced at Glaxo Wellcome Research and Development, Stevenage.

I would like to thank my supervisor, Dr. Charlie Hodgman, for his guidance and encouragement in this work, Alan Yates, for his invaluable programming advice, and all the staff in the Bioinformatics group at Glaxo Wellcome for their encouragement and support, and for providing a dynamic, friendly working environment.

Introduction

Feature tables are found in a number of biological data banks. Their use offers a simple but precise means of holding information about the structural and functional properties of a sequence. They are becoming increasingly important as a rapidly expanding knowledge base continues to demand clearer and more consistent organization of such annotation data.

However, there are a number of problems associated with using feature tables:

- (a) Currently, the formats of feature tables differ, sometimes considerably, between data banks. For a scientist to add their knowledge to a data bank directly, it is therefore necessary for them to convert it into a strict data bank-specific format which must be broad enough to encapsulate all the applicable categories of that knowledge.
- (b) As the number of entries in a feature table grows, the information space of that sequence becomes increasingly difficult to navigate.

Concerns such as these discourage scientists from making full use of sequence data banks, and prevent sequence data banks from reaching their full potential as dynamic information resources for the scientific community.

A computational tool that allows efficient visualization and editing of the information present in a feature table, and makes the specifics of a data bank format transparent to the user, could considerably narrow the gap between the scientist and the knowledge base they use and add value to. The rest of this chapter considers the requirements for a useful feature table editor, discusses user interface strategies in the context of existing feature table editors, and finally states the aims of the present work.

Requirements for a feature table editor

An ideal feature table editor might have the following properties:

- (a) full editing functions that allow the user to at least add, edit and delete features,
- (b) interactive viewers of the feature table, the sequence and the feature graphics,
- (c) saves the user's work in a useful format,
- (d) works through an interface with which the user is both comfortable and familiar.

Interface strategies

At present there is no official or de facto interface standard for viewing and modifying biological data. While the diversity of interfaces in bioinformatics software might in theory empower the scientist with choice, there is a recognized need in the bioinformatics community for compatibility both at the API and the user interface level. Significant developments have already been made towards standardization of the former (BioWidget Consortium, 1996). This section will briefly consider the objectives of an interface, describe and illustrate the two major interface strategies: the command interface and the graphical user interface, and discuss the most appropriate solution for the present work.

Interface principles

The user interface of a system is often the yardstick by which that system is judged. An interface that is difficult to use will, at best, result in a high level of user errors. At worst, it will cause the software system to be discarded, irrespective of functionality (Sommerville, 1992).

Any user interface must take into account the needs, experience and capabilities of the system user. A 'good' interface design might aim to:

- (1) use terms and concepts that are familiar to the anticipated class of user,
- (2) be appropriately consistent,

- (3) should not surprise the user,
- (4) include mechanisms to limit user error such as user confirmation of potentially destructive actions, and 'undo' facilities,
- (5) incorporate some form of user guidance.

More generally, an interface should empower the user, making them feel that goals and tasks are effortlessly achieved through the program's capabilities and tools (Head, 1997).

Command-line interfaces

Command-line interfaces require the user to interact with the system through the keyboard. A number of distinct interface types are described as 'command-line'. Here, the term will refer specifically to screen-oriented interfaces such as the UNIX text editor *vi*, in which the user typically initiates commands through 'hot keys' or by entering text.

NewFeatures

NewFeatures (Rice, 1996) is a feature table editor that allows the user to add, modify and delete features in a feature table and to perform minor editing of the sequence. Primarily for working with nucleotide sequences, the program can load and save files in EMBL, GCG, or EMNEW format from a local data bank or file system.

NewFeatures is part of the EGCG package and operates in a command-line environment similar to the VMS text editor EDT and the GCG sequence editing program SeqEd. Users are presented with a scrollable text area, which shows the feature table in data bank format, positioned above a scrollable alphanumeric diagram of the sequence (figure 1). Three translations are optionally shown below the sequence diagram. While in 'screen mode', a user can navigate the feature table and the sequence using single key presses. Editing and more advanced features are performed in a shell-like 'command mode'. The program tackles a number of problems associated

with editing sequences to which features and other information are attached, and provides a vocabulary of commands to do this.

```

X12345.Dat                                     NEWFEATURES

FH      Key      Location/Qualifiers
FH      CDS      262..1299
1 >> FT      /product="amidase"
FT      mRNA    <1..>2657
2 * FT

0 10 20 30 40 50 60 70
.....|.....|.....|.....|.....|.....|.....|.....|.....
AGCTTCGGTGGCAATGATGGCAGTCATGCTATCTCAGGCTCGCACCATGTGCTTTGCGGATTCGCGCGGATTTACA
S F R A N D G M H A I S G S H H V L S R S R R L H
A S V R M A C M L S Q A R T M C F R D R A D Y I
L C P E * W H A C Y L R L A P C A F A I A P I T

```

Figure 1. Screen shot from EGCG Newfeatures (Rice, 1996).

NewFeatures makes the most of its command-line environment but in doing so emphasizes the limitations of this type of interface. The user is allowed to examine any particular feature in the feature table but there is no way of viewing the complete dataset at one time. This may not be a problem for sequences with few features, but as the information space grows, the relationships between features become increasingly difficult to follow.

Displaying large information spaces in a way that does not overwhelm the scientist is a common problem in bioinformatics (Robinson and Flores, 1997). Effective solutions typically represent the data set at a coarser grain level through extensive use of graphics, making the command-line environment unsuitable in this respect.

Applications that use command-line interfaces have the advantage of being cheap to implement, requiring only an alphanumeric display, and of allowing rapid user interaction. However, command-line interfaces, particularly those that do not employ a menu-system, are often unsuitable for casual or inexperienced users. It is typically necessary for the user to learn a dictionary of command keys before they can perform even simple tasks. This produces an initially intimidating environment that often discourages new users from going further. What is more, as graphical interfaces

providing intuitive and friendly environments become more widespread, applications based on the command-line will suffer increasing prejudice from new users.

Graphical user interfaces

Graphical user interfaces offer a number of advantages over command-line interfaces. The field of human-computer interaction has shown that GUIs have design components that fundamentally match the cognitive abilities, expectations and limitations that a user brings to a system (Verplank, 1988). Commands and menu items that unfurl as needed limit the user's memory overload, while enhanced screen visuals that contrast the colour, font, shape and arrangement of objects can help a user to focus attention on the task at hand. Navigation by pointing and clicking a mouse provides intuitive full-screen interaction that does not require full keyboard knowledge or a memorized command vocabulary; while multiple windows allow the user view separate tasks non-exclusively. These properties can make applications based on graphical user interfaces easy to learn and use.

The most effective GUIs implement design principles that (i) empower users, providing interface flexibility, easy navigability, and allowing users to customize their environment; (ii) communicate visually, forcing attention to aesthetics that do not overwhelm or distract the user, meaningful representations that are clear and intuitive metaphors, and the consistency of graphics and functionality through the interface; and (iii) encourage interaction through point and click navigation that allows direct manipulation with immediate results, feedback, and forgiveness. These design concepts extend those propounded at the beginning of the section and may be illustrated with a brief discussion of the SeqLab package and its interface.

SeqLab

SeqLab is an X-Windows based application that combines the Wisconsin Package Interface (Genetics Computer Group, 1997) with the Genetic Data Environment (Smith et al., 1994). The Wisconsin Package provides a wide range of sequence

analysis tools, all of which are accessible through SeqLab. SeqLab integrates these with the single and multiple sequence editing tools of GDE. The package includes a feature table editor that allows the user to view and edit the feature tables of multiple sequences in a customizable graphical environment. A wide variety of data bank formats are accepted as input. Once the feature table has been modified, the user may save the updated sequence annotation in SeqLab-specific 'rsf' format.

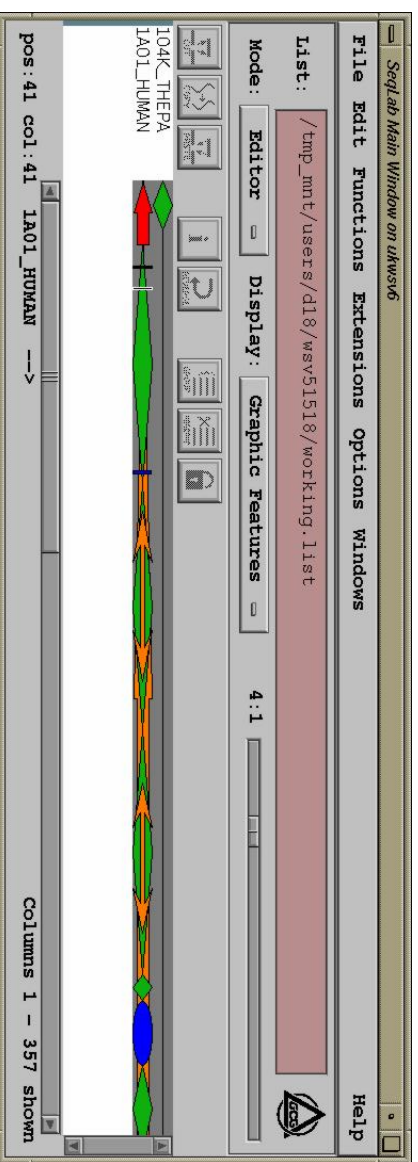


Figure 2. Screen shot from the feature table editor in SeqLab (Genetics Computer Group, 1997).

The SeqLab interface uses an arrangement of 'menubar' and 'choice' graphical components that are largely consistent with that found in PC, Macintosh, and other X-Windows applications (figure 2). The feature table editor displays the features of a sequence in a horizontal diagram where each feature is represented by a coloured shape threaded onto a thin line, the latter representing the sequence. The horizontal width each shape occupies relative to the thin line is proportional to length of the feature relative to that of the sequence. The colour and shape associated with each feature initially vary with different types of features but may be modified by the user at any stage. Double clicking on a feature in the diagram brings up information relating to that feature in a separate window. A slider is also present that allows the user to change the horizontal scale of the diagram.

These features empower the user, enabling them to customize their environment, and communicate visually, allowing the user to navigate a large information space at a coarse and fine grain level in a familiar environment. However, while the features may be represented by a number of often elaborate shapes, the diagram is so vertically

compressed as to make these features look grossly elongated and cramped. This economy of height may facilitate viewing of multiple sequences with few features, but it makes diagrams of one sequence with many features difficult to understand. Further, although it is inevitable that heavily annotated sequences will contain features that overlap, SeqLab's feature diagram does provide any way of visually clicking through multiple layers of features. The user's supposedly interactive view is restricted by this static representation.

Inconsistencies in and between different GUI models and conventions can produce a false sense of security in which users, at best, feel confused and frustrated and, at worst, waste time assuming non-existent functionality and make errors they cannot undo. SeqLab provides an intuitive environment that makes the user feel confident about adding, deleting and modifying features. But when an editing session is over and the user wishes to save their changes to a sequence annotation, the only way to do so is in a SeqLab specific format. The saved file typically contains the original data followed by text that describes the colours, shapes and positions of both original and new features. Users familiar with the open and save commands used in most other editing environments might reasonably expect the output to be in the same format as the input. The discovery that SeqLab does not support such functionality is likely to surprise and frustrate users, particularly if it is made after time spent editing a feature table, and may discourage their subsequent use of this editor.

JaMBW

Java-based Molecular Biologists' WorkBench, JaMBW (Toldo, 1997), is a suite of bioinformatics applets accessible through a web interface. Among the services JaMBW provides is a feature viewer that displays the features of a sequence in an interactive graphical format. Sequence and feature table input to the feature viewer must be in a custom format and is static, being accessed by the applet from its parent HTML page.

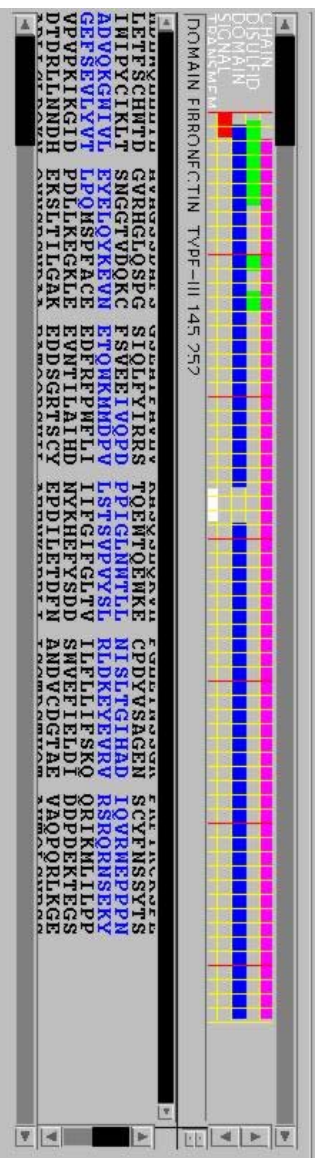


Figure 3. Screen shot from the Feature Viewer applet in JaMBW (Toldo, 1997).

The feature viewer is presented as an applet to be modified and integrated into a larger system, rather than as a useful application in itself. Although its presentation within JaMBW is cramped and makes it look awkward and potentially frustrating to use (figure 3), it may be resized and reoriented to more comfortable dimensions by modifying its parent HTML document. The feature table is represented as a grid where each row represents a feature type e.g. 'domain', while columns represent a length scale. Features are shown as coloured boxes stretched across each row. Clicking on a feature results in its details being shown in one scrollable text area while its position is highlighted in another containing the sequence.

The information presented in the feature viewer is static: it can only be input when the applet is initialized and cannot be modified. The limitations of static input, particularly when it is required in a custom format, restrict the usefulness of the applet to server-based systems where HTML documents are dynamically produced and applets dynamically loaded. The applet is not suitable for integration into the sort of dynamic client-side applications for which Java has considerable support.

Because the feature viewer is written in Java, not only can it run on potentially any platform but it also provides the user with a familiar interface by adopting the 'look and feel' of that platform's GUI components. Java will be discussed further in the implementation chapter.

The use of a grid effectively conveys both the relative positions and sizes of the features in the feature table. It is appropriate that essentially two-dimensional information is graphically represented in two dimensions. Separating the features

according to their key makes it easy for the user to consider the distribution of one type of feature without losing sight of the annotation as a whole, and allows global patterns in a potentially large information space to be quickly identified. The grid-based representation also overcomes some of the problems seen in SeqLab by not allowing the features of one type, e.g., disulphide, to be graphically swamped by those of another, e.g., domain.

The feature viewer does not, however, provide a means to distinguish overlapping or contiguous features of the same type, making it particularly unsuitable for nucleotide annotations that might include multiple overlapping alleles.

While colour can improve user interfaces by helping users understand and manage information complexity, it is a powerful tool that can easily be misused. Misuse of colour in GUIs is often through overuse of vivid colours that disturb the user, or the extensive use of 'bad' colour pairings that cause eye-strain (Sommerville, 1992). If anything, the feature viewer applet has the opposite problem. The user is presented with a feature diagram that uses a range of 'washed out' colours and white text on a light background. Combined with the fact that there is no means of customizing the colours, this may prevent the applet from reaching its full potential for many users, making its intuitive diagrams difficult to read and its graphics lack appropriate impact.

Typically, the more extensive, interactive and dynamic the use of graphics in an interface, the greater the computational overhead (Sommerville, 1992). The need to optimize graphics to avoid user frustration becomes acute for programs written in non-compiled platform-neutral binaries such as Java and, to a lesser degree, Tcl/Tk. JaMBW's feature viewer suffers from slow, inefficient and often incomplete repaint procedures. While this is may be a relatively minor detail, it is one that slows user interaction and might even cause the user to unfairly lose confidence in the program's robustness.

Aims

The aim of the work presented here is to develop a computational tool that enables a user to visualize and edit a feature table. Specifically, the program is required to:

- (a) receive input in a data bank format;
- (b) work through a standard graphical user interface;
- (c) allow the user to add, modify and delete features in the feature table;
- (d) provide interactive customizable graphics representing the features and the sequence;
- (e) provide the user with an interactive view of the feature table;
- (f) save results in a useful format.

Implementation

A feature table editor program was written in Java version 1.02 (Sun Microsystems Inc., 1994), over a 12-week period. The program, named 'J-Features' (Java Feature Table Editor), was designed to run as an applet from a World Wide Web page.

This chapter will first discuss the choice of language and platform, then provide a brief introduction to Java and its associated terminology. The remainder of the chapter will describe the design of the program, first in general and then in specific terms, followed by a user perspective of the program and overview of its functionality.

Languages and platforms

Graphical user interfaces can differ greatly between platforms not just in terms of their 'look and feel' but also in terms of their programming interface or 'toolkit'. Porting a program written for one computer architecture to another can be extremely difficult and sometimes requires a complete rewrite of the code (Schildt, 1995; Sommerville, 1992). This problem is made worse both by the typical complexity of toolkit interfaces and by the fact that some platforms, e.g., PCs that use Microsoft Windows, regularly update and change their windowing system. Some platform-specific languages exist that simplify and genericize the toolkit interface. These include Visual Basic (Microsoft, 1997) for the PC, and Tcl/Tk (Ousterhout, 1994) for the X-Windows/Motif toolkit on UNIX machines.

Java (Sun Microsystems Inc., 1994) gets round these problems of portability by introducing a run-time layer between the specifics of the platform and the program code. This layer is called the Java Virtual Machine (JVM) and is implemented by most Internet browsers. Since most platforms support Internet browsers, programs written in Java are effectively platform-independent. The Java specification includes an 'abstract windowing toolkit' (AWT) that provides a consistent programming interface to the GUI toolkit of the native platform. Its support for graphics is therefore strong and robust.

Since the feature table editor is intended for scientists working on any platform and is required to incorporate extensive graphics and GUI features, Java was considered the most appropriate language for the task.

The next section provides a fuller discussion of Java and introduces concepts and terms that will be used extensively in the remainder of this work.

Java

Java is an object-oriented language that derives many of its conventions from C and C++. Java differs from C and C++ in a number of ways that variously make it slower, more portable, simpler and more robust. This section will briefly describe some of the key features of Java relevant to this report.

Java is a simple language. When Java was designed, its syntax was kept small and familiar by taking the features available in C and C++ and then removing those that were rarely used, unnecessary or led to bad programming practice (Flanagan, 1996). Examples of these are unions, structures and the 'goto' statement respectively. Importantly, Java does not use pointers. Arrays and strings are considered objects, and referencing and dereferencing are handled automatically. The exclusion of these features, particularly of pointers, makes Java more consistent and considerably easier to program than C and C++.

Unlike C++, Java is object-oriented from the ground up. This has been cited as both an advantage and a disadvantage of Java (Brackeen, 1997). Object-oriented programming represents a paradigm for writing 'good' programs for a set of problems, and languages that support it, by implication, support the techniques of data hiding, data abstraction, inheritance and run-time polymorphism (Stroustrup, 1991). While it is outside the scope of this work to fully describe object-oriented programming, it is worth mentioning some of the advantages of OOP and also how Java differs in its support for this type of programming from C++.

In an object-orientated design, a system is considered in terms of a group of entities, or objects, that interact with one another (Sommerville, 1992). Because the objects are independent, object-oriented design can produce modular architectures in which components can be easily replaced, added or reused in other systems. The development of platform-neutral widgets has become a priority in the bioinformatics community and Java is considered an ideal implementation language for this (Robinson & Flores, 1997).

Java has a different inheritance model from C++, which supports Multiple Inheritance (MI), where a class can have more than one superclass. Object-oriented programming languages are split on the issue of whether or not MI is more trouble than it is worth. Certainly, the use of MI opens up a 'Pandora's box' of complexity and ambiguity (Meyers, 1992). Java provides a more elegant solution in the form of 'interfaces'. Interfaces are non-implemented API definitions and are analogous to C++ abstract classes in which all member functions up the class hierarchy are pure-virtual. A Java class can inherit from, or 'extend', one superclass, but can implement the methods of any number of interfaces. A class that does this can be dynamically cast to any of the interfaces it implements. The interfaces can therefore be used like 'wrappers' without incurring the confusion of multiple base class implementations.

Unlike C and C++, Java is not compiled to machine code, but to an optimized 'byte-code' format that the Java Virtual Machine interprets in a platform-dependent manner at run-time. Because Java is interpreted, it is presently much slower than its compiled relatives. It can be argued, though, that what Java loses at run-time, it makes up for in development and porting time. This argument is made stronger by the fact that Java is significantly more robust than C or C++. Java not only dispenses with pointers, but also automatically handles garbage collection, preventing memory leaks, anonymous memory referencing, and other pernicious bugs (Flanagan, 1996). Java also implements an automatic reference counting system similar to 'smart pointers' in C++, facilitating code-optimizing techniques such as lazy evaluation (Meyer, 1996). Its strong typing and extensive use of exception handling allows for faster debugging at compile- and run-time, respectively. The combination of these features enables rapid

prototyping, easy experimentation and more confident use of advanced programming techniques, such as multithreading.

Java programs can be written as local applications or as distributed applets. Presently, most user interaction with Web pages is performed by Common Gateway Interface (CGI) scripting, which requires resources on the server. Because an applet is downloaded from a Web page onto a local machine, Java moves processing to the client, freeing server resources. Applets offer a high level of security because they run in a protected environment. In Java version 1.02, this means that, unlike Java applications, Java applets do not support local file access. This potential limitation of applets is resolved in Java version 1.1, where client-side file access is allowed, but subject to a number of security checks (Flanagan, 1997). Because Java 1.1 is not yet fully supported by common Web browsers, J-Features was written in Java version 1.02.

Java includes many other important features, particularly those that allow applets to seamlessly use and integrate disparate networks. It also has some problems (Bracken, 1997). Java will be considered further in the discussion chapter of this report.

Design of J-Features

J-Features has a modular design. The relationships between its principal components are represented in figure 4. Modules have been made generic beyond the scope of the program structure. For example, parent objects communicate with child modules through child-specific interfaces they implement.

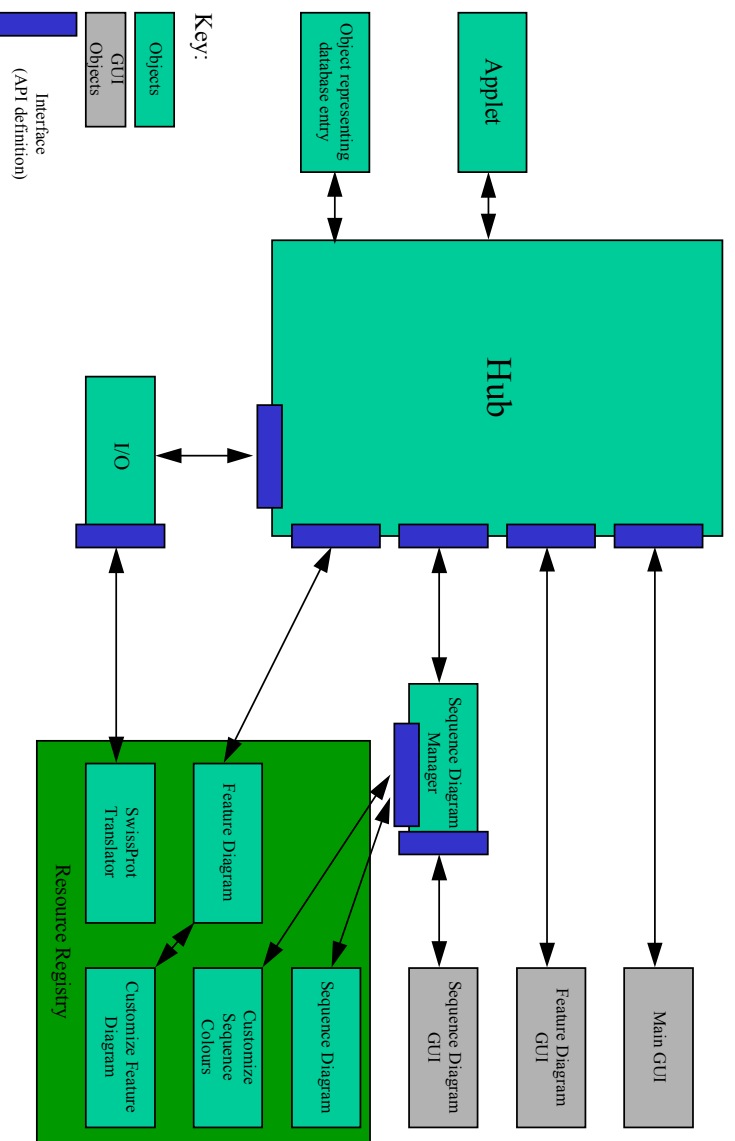


Figure 4. Schematic diagram of the structure of the J-Features program. Arrows represent the path of communication between the objects.

The feature table editor can receive input and generate output in SwissProt (Bairoch and Boeckman, 1994) format, doing so through a 'copy-and-paste' I/O model. The objects for both the data bank format and the I/O model communicate with each other and the rest of the program through generic interfaces.

The program has two main graphical widgets that represent the information space of the data bank entry: the feature diagram and the sequence diagram. The feature diagram represents the features as coloured shapes strung on a horizontal line, the latter representing the sequence. The sequence diagram represents a sequence of amino acids or nucleotides as a scrollable string of coloured letters. The widgets are separate from the GUI macrocomponents that frame them and facilitate their manipulation.

To minimize computational and memory overhead, the program uses a shared information resource, FTED_ResourceRegistry. The resource registry allows information used in one part of the program to be altered in a controlled manner by

another part of the program. Its implications, with regard to object-oriented design principles, will be considered in the discussion chapter.

In addition to its 56 custom classes, J-Features used classes from the following packages: Java Development Kit version 1.02 (Sun Microsystems Inc., 1994), Graphic Java Toolkit version 1.0 (Geary and McClellan, 1997), JGL (ObjectSpace Inc., 1997), and Symantec (Symantec, 1996). Appendix 1 provides a brief description of the classes written for J-Features, while appendix 2 shows the class hierarchy.

J-Features: A user perspective

This section will describe the user interface to the feature table editor applet, J-Features.

Starting off

The J-Features feature table editor is a Java program that uses multiple independent frames to simulate an editing environment. It differs in this respect from many applets that contain all their GUI features within one Web page. When a user accesses the parent Web page through Netscape, or another Java enabled browser, they are confronted with a single large red 'Start' button, under the title of the program (figure 5).

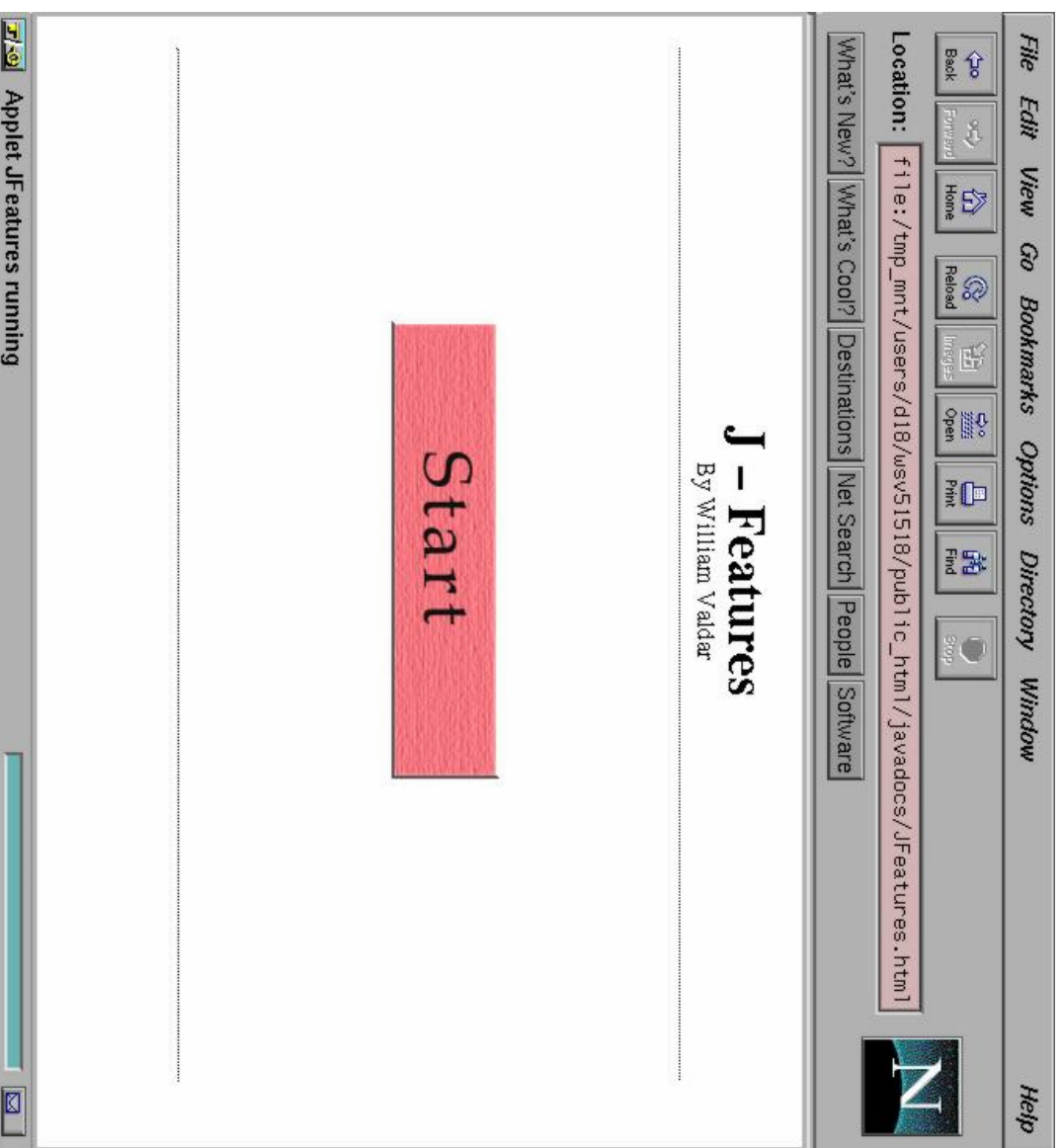


Figure 5. Screen shot from the J-Features applet: Web page. The Web page from which the editor is called, as viewed through Netscape version 3.01.

When the user clicks this button, two frames appear: one with a file menubar over a title logo; the other, entitled 'feature diagram', with a picture of a horizontal line above a panel of buttons (figure 6). To begin a session, the user selects 'Open' from the file menu. A new frame, entitled 'Database Input' appears (figure 7). The user pastes information in SwissProt format into the large text area of this frame and then clicks 'Accept'. The information may constitute anything from a full SwissProt entry with HTML tags, to a small untagged section of such an entry. J-Features extracts what information it can from the input, assuming defaults for information not given.

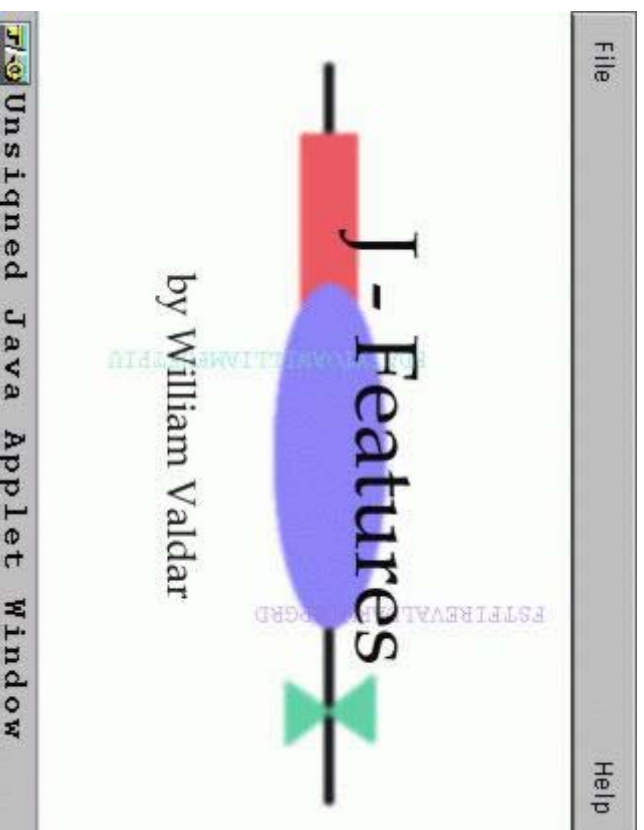


Figure 6. Screen shot from the J-Features applet: title frame. The title frame contains a menu system from which I/O operations can be initiated.

```
DR EMBL: <A HREF=#wgetz?-id+20mZAI7379P+-e+Hembi-acc/x
DR EMBL: <A HREF=#wgetz?-id+20mZAI7379P+-e+Hembi-acc/l
DR PIR: <A HREF=#wgetz?-id+20mZAI7379P+-e+Hpir-acc/S141
DR HSSP: <A HREF=#wgetz?-id+20mZAI7379P+-e+Hhssp-!d;P0
DR MIM: <A HREF=#wgetz?-id+20mZAI7379P+-e+Hmim-!d;14
DR PROSITE: <A HREF=#wgetz?-id+20mZAI7379P+-e+Hprosite-
KW MHC_I; TRANSMEMBRANE; GLYCOPROTEIN; SIGNAL; POLYMC
FT SIGNAL 1 24
FT CHAIN 25 365 HIA CLASS I HISTOCOMPATIBILITY
FT 25 365 ALPHA CHAIN A-1.
FT DOMAIN 25 114 EXTRACELLULAR ALPHA-1.
FT DOMAIN 115 206 EXTRACELLULAR ALPHA-2.
FT DOMAIN 207 298 EXTRACELLULAR ALPHA-3.
FT DOMAIN 299 308 CONNECTING PEPTIDE.
FT DOMAIN 299 308 CONNECTING PEPTIDE.
FT TRANSMEM 309 332
FT DOMAIN 333 365 CYTOPLASMIC TAIL.
FT CARBOHYD 110 110 BY SIMILARITY.
FT DISULFID 125 188 BY SIMILARITY.
FT DISULFID 227 283 BY SIMILARITY.
FT VARIANT 33 33 F-> S (IN A*0102).
FT VARIANT 41 41 R-> S (IN A*0102).
SQ SEQUENCE 365 AA; 40846 MW; 8EB80E9E CRC32;
MAVMAPRTLL LLLGALALT QTWAGSHSMR YFTSVSRPG RCEPR
DSDAASQKME PRAPWIEDEG PEYWDQETRN MKAHSQTDRA NLC
IMYCCDVGPD GRFLRCYRQD AYDCKDYVAL NEDRWSWTAQ DMAA
RVLEGRQVD CLRVLNCKR ETLQRTDPPK THMTHHPISD HEATL
WQRDGEDQTD DTELVELTRPA GDQTFQKMAA WVPSPCEEQR YTC
```

Figure 7. Screen shot from the J-Features applet: input frame. Users enter input in the central text area of this frame.

Upon accepting input, the feature diagram is updated, showing a number of rectangles and egg-timer shapes along its horizontal line (figure 8). A new frame, entitled 'Sequence Diagram' and containing a scrollable image of the sequence above a number of controls, also appears (figure 9).

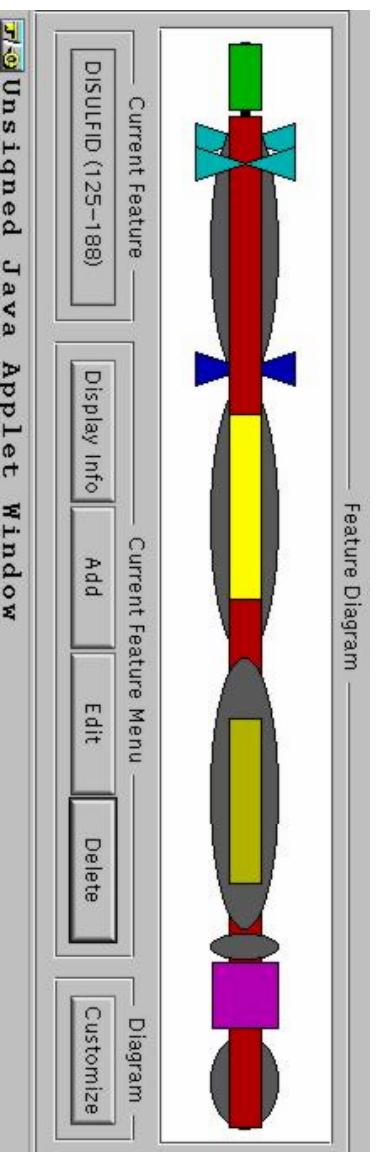


Figure 8. Screen shot from the J-Features applet: feature diagram. The feature diagram represents the features as interactive coloured shapes sitting on a horizontal line, which represents the sequence.

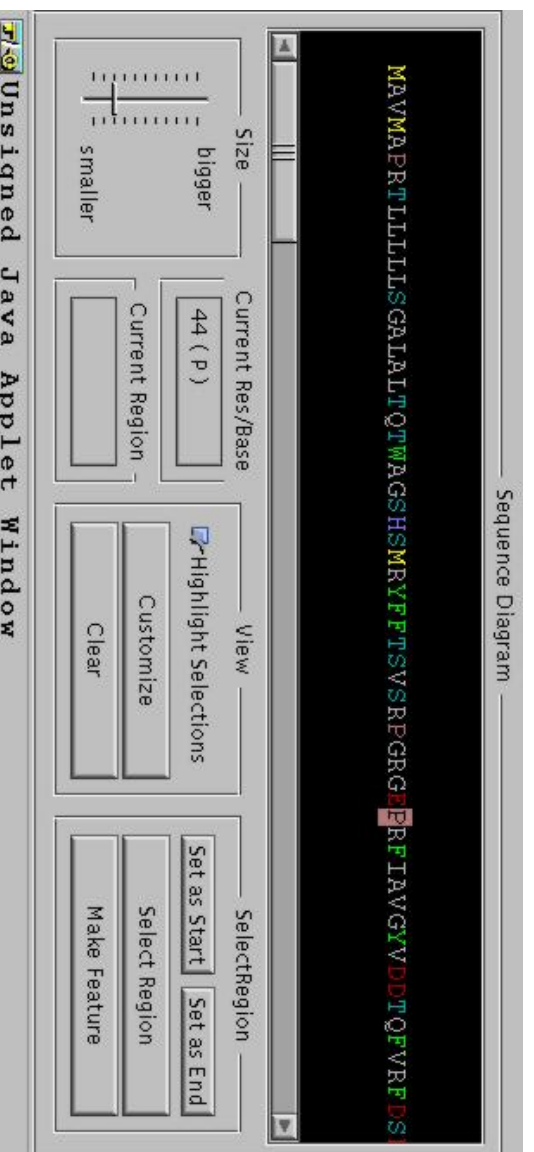


Figure 9. Screen shot from the J-Features applet: sequence diagram. The sequence is represented by a scrollable, interactive, multicoloured string of letters.

Working with the feature diagram

Initially, all the features on the feature diagram are represented by either low dark-red rectangles or tall dark-red egg-timer shapes (figure 8). Features too small to be easily seen as boxes are represented by egg-timer shapes, the 'neck' of an egg-timer coinciding with the centre of that feature's locus. Clicking on a shape causes it to be highlighted, and results in the type and position of the feature it represents being displayed in the 'Current Feature' box situated below the diagram (figure 8). Often, one feature is obscured by other features that overlap or cover it. The user can therefore 'click through' any number of layers to select the feature they want (figure 10). If the parent frame is reshaped or maximized by the user, the feature diagram

stretches to fill the space available. When stretched, features previously depicted as egg-timers are represented by boxes if necessary.

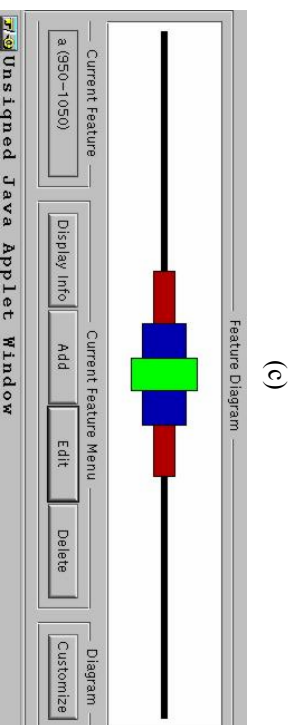
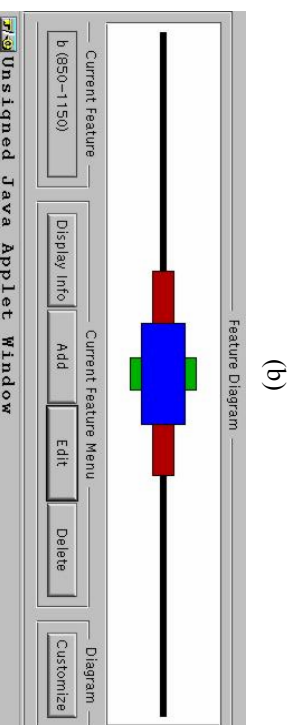
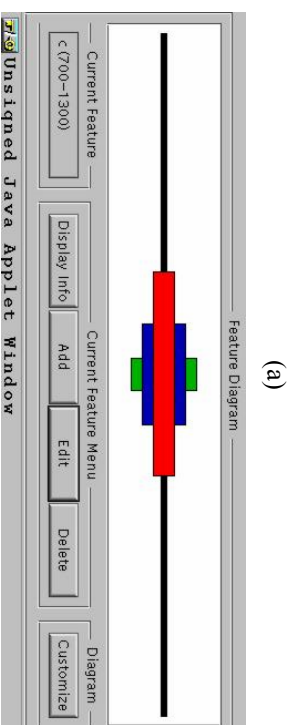


Figure 10. Screen shots from the J-Features applet: selecting features on the feature diagram. (a), (b), and (c) show three features that all overlap at one locus. Clicking on this locus once selects the red feature (a); clicking a second time selects the blue feature (b), and a third selects the green feature (c).

The components below the diagram are separated into categories relating to their function. Under the heading of 'Current Feature Menu' are the buttons 'Display Info', 'Add', 'Edit' and 'Delete'. Clicking 'Display Info' brings up a read-only text frame displaying the type, position, qualifiers and comments relating to the selected feature (figure 11), and highlights, underlines and scrolls to the feature's residues in the sequence diagram (figure 12).



Figure 11. Screen shot from the J-Features applet: feature information frame. This frame shows information about the feature in no particular format.

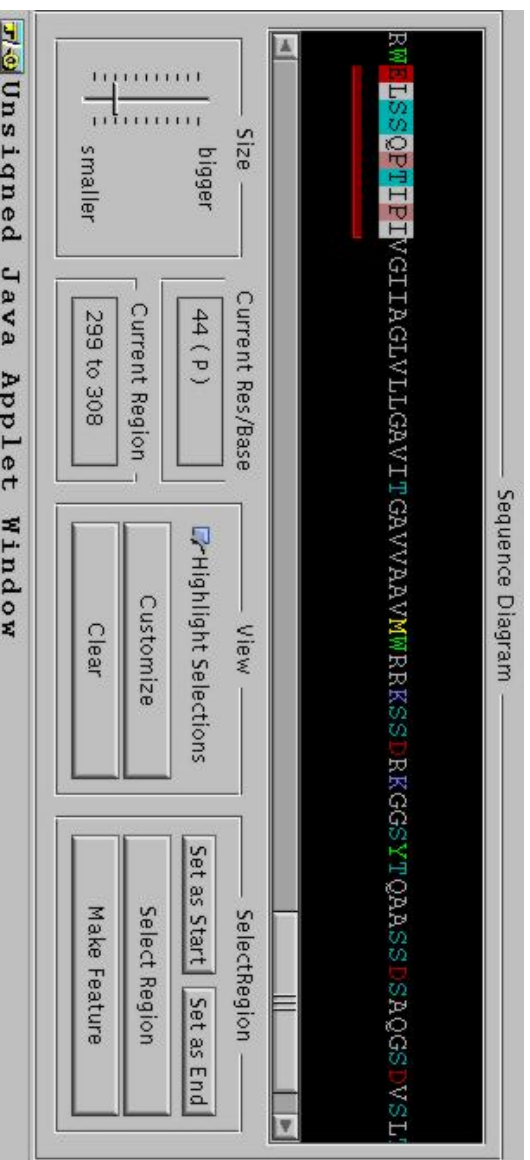


Figure 12. Screen shot from the J-Features applet: a highlighted region in the sequence diagram.

Clicking 'Delete' brings up a 'Question' dialog box that asks the user to confirm deletion of the selected feature (figure 13).



Figure 13. Screen shot from the J-Features applet: dialog box requesting confirmation of deletion of a feature.

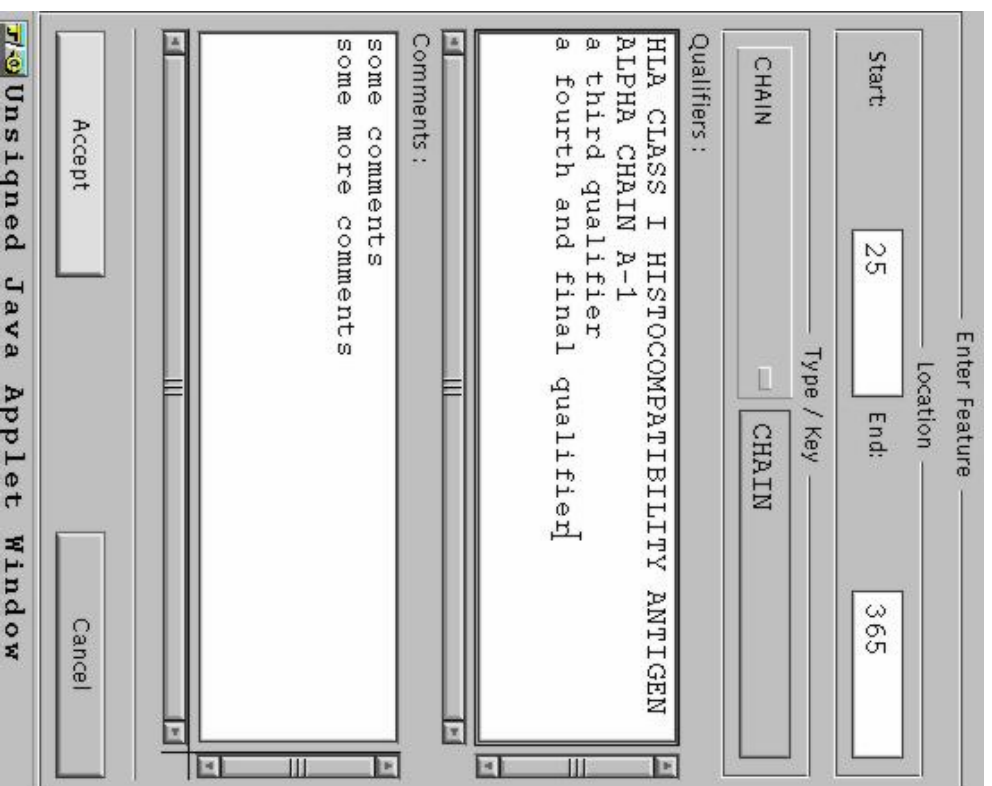


Figure 14. Screen shot from the J-Features applet: feature input form. This form allows the user to input information about a new or existing feature. The figure shows a feature input form with some example input.

Clicking 'Add' brings up a frame containing a of text boxes in which details about a new feature can be entered (figure 14). In the 'Type/Key' box, the user can choose one of the standard names for a SwissProt feature type or a custom name. If the information about a feature is incomplete or contradictory, e.g., the feature's end precedes its start or no type name is given, clicking 'Accept' will bring up a dialog box that informs the user of their mistake and, when dismissed, will allow them to continue editing the information.

Clicking 'Edit' in the 'Current Feature Menu' brings up a same input form used in 'Add', completed with the selected feature's information.

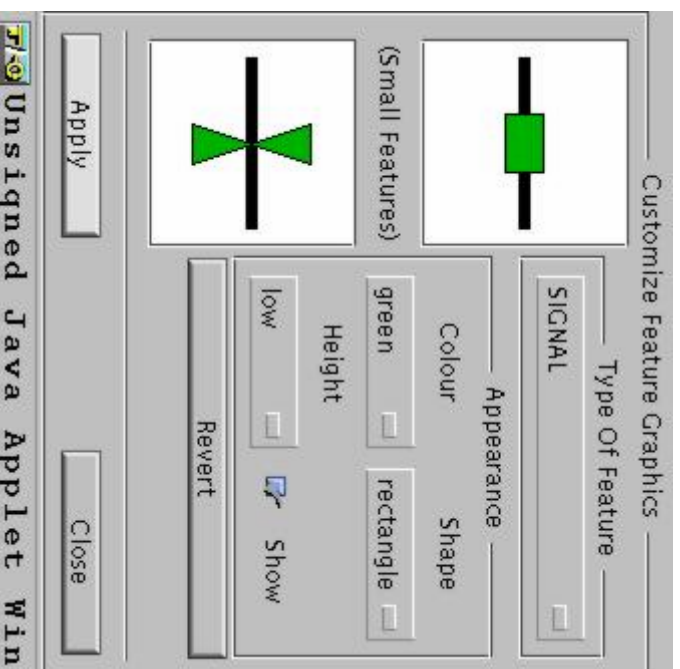


Figure 15. Screen shot from the J-Features applet: customize features frame. This frame allows the user to customize the appearance of features on the feature diagram.

Clicking 'Customize' in the 'Diagram' box under the feature diagram brings up a new frame that allows the user to customize the appearance of the features (figure 15). Feature types can be assigned different colours, shapes and heights. The interface allows the user to preview possible feature graphics without applying them in two highlight-able test boxes. If the information on the diagram is overwhelming or excessive, the user can choose to not show types of feature. Clicking the 'Apply' button causes the feature diagram to be updated. The information space of the feature table can therefore be made more navigable through user-defined schemes for feature colour, shape and size.

Working with the sequence diagram

The frame entitled "sequence diagram" contains a scrollable image of the sequence above a number of control panels separated into categories (figure 9). Residues on the diagram are coloured according to the scheme used in Cn3D (Hogue, 1997). Clicking on a residue causes it to be highlighted and results in the number and identity of that residue being displayed in the 'Current Res/Base' box.

The font size of the letters, and hence the size of the sequence, can be adjusted by moving the slider widget in the 'Size' box.

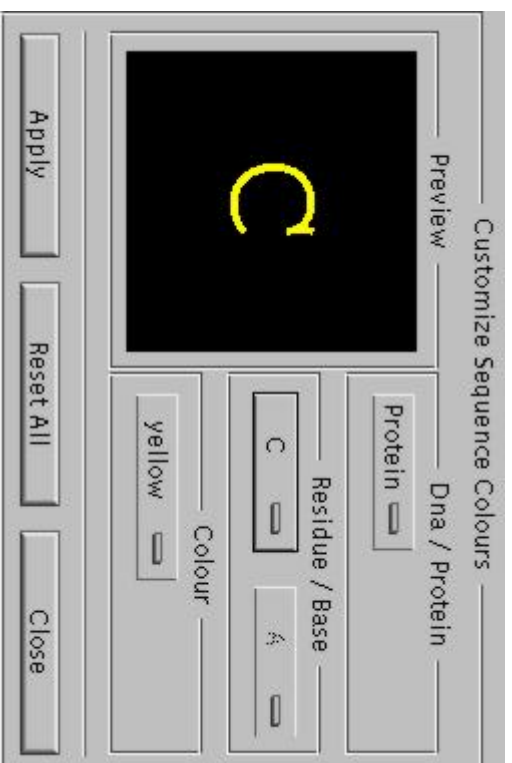


Figure 16. Screen shot from the J-Features applet: customize sequence colours. This frame allows the user to customize the colours used in the sequence diagram.

The 'Select Region' box provides tools to select regions of the sequence and assign new features to those regions. A region is selected by clicking on a residue, setting it as the start, clicking on another -- or the same -- residue, setting that as the end, then clicking the 'Select Region' button. As with the 'Display Info' function described above, this causes all the residues within the region to be highlighted and clearly underlined, as well as displaying the start and end of the selected region in the 'Current Region' box. Clicking on 'Make Feature' brings up the input form described earlier with its start and end boxes filled.

Clicking on the 'Customize' button brings up new frame entitled 'Customize Sequence Colours' (Figure 16). This interface provides tools that allow the user to change the colour scheme of the sequence. It is similar in its operation to the customize tool described for the feature diagram, and includes a selectable preview graphic.

re-enter their preferences for each annotation they edit. J-Features returns to its default settings only when it is closed, by clicking 'Close' from the file menu, and re-opened, by pressing 'Start' on the parent Web page.

In principle, multiple copies of J-Features can be run from the Web page at once. However, the use of the FTED_ResourceRegistry makes this likely to cause minor bugs that, while not affecting the feature table, may surprise the user. The practice has therefore been prohibited in the code by implementing a SingleChildRestriction interface with respect to the objects JFeatures and Hub.

Discussion

The design of an interactive feature table editor that uses graphics, graphical user interfaces, and text requires a combination of techniques and concepts from a broad range of software engineering disciplines. These include: text processing; design and optimization of graphics; intuitive interface design; procedural processing; event-driven processing; and all the usual challenges of designing a medium-sized and stable application. The limits of what can be achieved in these areas are continually being pushed back, as more and more advanced, interactive and intuitive word processors, graphics packages and games are developed by a relentlessly competitive and expanding software industry. User expectation of what is considered 'acceptable', let alone 'good', rises in concert. Arguably, a user-centred application that expects to have any kind of lifetime must either provide a high level of functionality in some or all of these areas, or be flexible enough to allow such functionality to be simply added at a later stage. Software development in biology is more specialized and not subject to quite the same pressures as software development in general, but the distance between the two narrows as the role of bioinformatics becomes more prominent.

The design of J-Features required a practical approach that could deliver generic, robust and useful software in a short time. For this reason it was most appropriate for J-Features to be designed as a prototype that could be easily modified and extended to include a wide range of further functionality.

The program was designed to have modular structure. Child-parent dependencies were minimized where possible to provide a high degree of encapsulation throughout the program. It was intended that reduction of J-Features to groups of reusable components should not be a problem at any level.

Objects that represented macrocomponents communicated with their parents through interfaces rather than 'hard' class types. For example, the object that managed input and output communicated with an 'IOListener' interface. In terms of how that object functioned, the identity of the object that implemented the interface was not

important. The use of word 'listener' in the program's interfaces is similar but not identical to its use in the Java 1.1 event handling system. This unfortunate clash of terminology would benefit resolution in a Java 1.1 implementation for J-Features.

The code for handling events within the larger GUI macrocomponents was completely separate from the code for generating their on-screen image. The latter, being considered largely cosmetic and most likely to change, were given their own classes and separated from the core program. The presentation of J-Features can therefore be changed without having to change the underlying computational system.

I/O model

The specifics of network protocols typically differ between commercial and public domain environments. Despite Java's networking capabilities, it was not therefore considered appropriate for the prototype to implement a network based input and output, since this may have compromised its portability at an early stage. Further, because applets written in Java 1.02 do not support client-side file access, 'cutting and pasting' was the most suitable prototype I/O model. The modular nature of the program allows the I/O model to be replaced with relative ease. Local file access, which is supported in Java 1.1, or environment-specific network operations could therefore be implemented in later versions of the program.

J-Features was designed to enable input and output in data bank format. However, an ideal feature table editor might be fluent in a large number of data bank formats. The prototype was therefore designed to parse input and save output in SwissProt format (SwissProt, 1994), while being flexible enough in its structure to allow for other formats to be added at a later stage. Flexibility here, as in the rest of the program was achieved through generic interface types. Specifically, objects that convert information between internal and data bank formats must implement the minimal `ObjectFlatFileConverter` interface.

However, the I/O aggregate within the prototype was not as complete or useful as it could have been. While the prototype design facilitates use of alternative data bank formats, it does not provide an infrastructure for handling multiple ObjectFlatFileConverter modules. The support of multiple data bank formats would add another level of complexity to J-Features, in terms of both the user interface and the structure of the I/O module. For example, it is not obvious whether the format of file input should be selected by the user or determined by the program. Further, the likely implementation of functions that save one feature table in multiple data bank formats would not be compatible with the present use of format-specific subclasses for internal data storage.

Graphical widgets

As the quantity and complexity of information in biological data banks increases, so too does the requirement for visual techniques that make this information navigable. There is particular emphasis in bioinformatics on realizing these techniques as reusable object-oriented widgets (Robinson and Flores, 1997; BioWidget consortium, 1996). Ideally, widgets are software components with robust, complete interfaces that are largely independent of the semantic nature of the data they display or process. Specialization is provided by the rapidly prototyped applications that bind these widgets together (Searls, 1995). Indeed, applications of this type have the dual purpose of addressing a specific set of problems, while illustrating novel application of the widgets they incorporate.

J-Features represents the information space of a sequence annotation through the feature and sequence diagrams. These components dynamically represent the data, but, in accordance with good design practice (Sommerville, 1992), do so through regularly updated information 'mirrors' rather than through direct references to the core information itself. This independence from the rest of the program means that, at least within the semantic constraints of the feature table editor, they are encapsulated, reusable, widget-like entities. However, there are several reasons why their present

implementation makes them unsuitable as widgets for incorporation into other programs.

Feature diagram

The model for visualizing the features of the sequence described here is similar to that used by SeqLab, but has the advantage that multiple layers of features can be accessed in an intuitive and highly interactive manner. To counteract the slow execution of Java's interpreted code, the graphics procedures in the DiagramOfFeatures class were optimized. Despite increased speed and interaction, however, the one-dimensional representation used may not be the most efficient approach to visualizing complex, highly annotated sequences. The fact that features can be obscured by each other at all means that important information is hidden from the user, even when the complexity of the annotation is moderate. One solution might be to provide an additional view representing the annotations for a focus region. This alternate representation could use the expansive two-dimensional approach of JaMBW's feature viewer, but have each feature with its own row so that multiple layers can be viewed at once. The overlap problem in the original diagram could be reduced by making features that are not selected slightly transparent with respect to the features they obscure.

Data abstraction at the programming interface allows widgets to be smoothly incorporated into a variety of applications. However, the current implementation of the feature diagram module, specifically the DiagramOfFeatures API, requires objects of J-Features-specific classes as parameters. J-Features could be extended to remove such semantic dependencies.

The feature diagram is highly customizable and, with the addition of more shapes, colours and other graphics, has the potential to be even more so. A user might invest considerable time and effort in optimizing the scheme of colours and shapes for their purposes. The module would be more useful if it allowed the diagram to be output as a graphic, for example, in postscript format. It could even include a facility for saving the user's preferences.

Sequence diagram

The sequence diagram module has a robust and generic programming interface. The use of colour and interaction empowers the user and emphasizes the diversity of the bases or residues. However, its representation of the sequence, combined with how it implements that representation, makes its use limited as a visualization technique. Displaying the sequence along a single line is useful for focusing on particular regions but frustrating if the user wishes to view the whole sequence, even when the font-size has been reduced. The sequence diagram expands to fill the space available, allowing the user to adapt the program to the dimensions of their workspace rather than the other way around. But extending the diagram vertically adds nothing to the display, and merely emphasizes squandered space below the characters of the sequence. Optional 'wrapping' of the sequence, in which the character string is continued on a series of new lines, may facilitate the diagram's support for context. Further, although the position of a residue or base can be determined by clicking on it, the addition of a numbered scale that ran parallel to the sequence would make the diagram considerably easier to navigate.

Navigating the sequence is made more cumbersome by poor scrolling. Although the graphics of the sequence had been optimized, the repaint procedures within the viewport of the scrolling window are still slow and inefficient, and could benefit from double buffering and clipping techniques.

Rubberbanding, in one form or another, is a common, well recognized, and intuitive way of selecting multiple items in a graphical display. The technique was not used in the sequence diagram because, while selection of multiple residues or bases by simply dragging the mouse over them may be useful for marking small regions within the viewport, it would make selection of larger regions, such as domains, awkward. A more robust, if slightly less intuitive, method is described in which the start and end positions of a block selection are chosen by separate events.

A number of other features could be added to the sequence diagram to make it more useful and intuitive. Selecting a residue could optionally cause other information, such

as a summary of which features that residue is part of, to be displayed. An interactive diagram that represents a sequence has the potential to integrate at least minor sequence analysis functions, such as pattern matching. The module could be extended to provide an infrastructure for implementing any number of analysis functions.

A more advanced sequence diagram might allow multiple sequences to be displayed together in a predefined alignment to provide another level of context during editing.

User interface

The J-Features user interface was designed to be attractive, intuitive and, where possible, familiar. As with most principal parent frames, the J-Features title frame contains a menu bar with 'file' and 'help' operations. The other GUI macrocomponents provide services for which there are no well defined interface conventions. Menu systems are useful when a relatively large number of commands fall into well-defined categories. The services provided by the macrocomponents are specialized, and, in the present implementation, can be reduced to a minimal but complete set of functions. Interfaces based on the control panel metaphor, rather than on menu systems, were thus considered most appropriate in this case.

The prototype specification uses multiple frames to emphasize the logical separation of different parts of the program. From a user perspective, this may not be the most efficient type of interface. While applications that use multiple frames can provide the user with a high degree of control over their work space, this model can confuse and overwhelm those with smaller displays. For this reason, the prototype hides windows that allow input, output, and customization of the graphics until they are requested by the user. The separation of the GUI code from the core program described above allows an alternative model, such as one based on a single web page, to be easily implemented.

Allowing the user to undo their mistakes is one of the most important principles of interface design and perhaps one of the most awkward to implement (Sommerville,

1992). There was not time to provide robust support for undo functions in the prototype. The present implementation of J-Features does, however, force the user to confirm potentially destructive actions and so reduces the frequency of such errors.

The interface was designed to be as simple as possible, but it does not provide any on-line help or information as to what the program actually does. The prototype code does not provide an infrastructure that supports real-time interactive help functions. These were considered as secondary to its core functionality and, given the speed of Java, would have required considerable time to implement efficiently. Help could be more appropriately provided through HTML documentation accessible from the parent web page.

Extending J-Features

Before the prototype can be extended, several problem areas must be addressed. The code incorporates a high degree of modularity, but this modularity is not consistent. For example, the activities of the DiagramOfSequence and DiagramOfSequenceGUI are coordinated by the central Hub object through an intermediate DoSManager object, whereas the more complicated DiagramOfFeatures and DiagramOfFeaturesGUI interact with Hub directly.

J-Features currently uses a shared data area, the FTED_ResourceRegistry, that provides a sink of information for number of objects occupying different locations in the parent-child hierarchy. This compromises object-oriented design principles. The use of shared data areas increases overall system coupling (Sommerville, 1992). If objects share variables, rather than communicate by exchanging messages, modifications to one such object can affect the actions of another in an implicit and unpredictable manner. Such shared data areas should therefore be used judiciously, and future versions of J-Features may benefit by reconsidering the role of the FTED_ResourceRegistry.

J-Features could be extended both in terms of providing more robust support for existing functionality and in terms of providing a wider range of services for a larger

set of problems. In addition to those discussed already, a future version of J-Features might incorporate the following improvements and extensions.

- (a) The input and output model could be extended to incorporate data bank and database connectivity and, with a Java 1.1 implementation, local file access. In the revised model, the user would input the name of the data bank entry or select it from a list, and the program would appropriate the data itself.
- (b) Multithreading could be used to make I/O and graphical operations more efficient.
- (c) Support for other, more complicated, data bank formats could be provided as could support for multiple sequences.
- (d) The J-Features programming interface could be modified to that of a Java 1.1 'bean'. As a bean, J-Features would become an embeddable, reusable software component that could be rapidly incorporated into another application (Flanagan, 1997).

Conclusion

The program described in this report provides an interactive web-based environment that allows a scientist to add their knowledge to a sequence annotation, while overcoming some of the problems encountered by contemporary feature table editors. Though robust in its current implementation, J-Features was written as a prototype that could be extended to produce a platform-independent feature-table editor incorporating a wide range of additional functionality.

References

- Bairoch A, Boeckmann B (1994) SwissProt The SwissProt protein sequence data bank - current status. *Nucleic Acids Res.* 22 : 3578- 2580.
- BioWidget Consortium (1996) home page, <http://goodman.jax.org/projects/biowidget/>.
- Braken M (1997) Battle for the House of Java. *Information Week 3* (16/9/1997): 20-24.
- Flanagan D (1996) *Java in a Nutshell: A Desktop Quick Reference for Java Programmers*. O'Reilly & Associates, Inc.
- Flanagan D (1997) *Java in a Nutshell, Second Edition: A Desktop Quick Reference*. O'Reilly & Associates, Inc.
- Geary DM, McClellan AL (1997) *Graphic Java: Mastering the AWT*. SunSoft Press/Prentice Hall.
- Genetics Computer Group, Inc. (1997) *Wisconsin Package Documentation*. <http://www.gcg.com/products/documentation.html>.
- Head AJ (1997) A question of interface design: how do online services measure up? *Online May/June p20-29*.
- Hogue C (1997) *Cn3D* (See in *Three-Dee*). <http://www.ncbi.nlm.nih.gov/Structure/cn3d.html>.
- Meyers S (1992) *Effective C++: 50 Specific Ways to Improve Your Programs and Designs*. Addison-Wesley Publishing Company, Massachusetts.

- Meyers S (1992) *More Effective C++: 35 New Ways to Improve Your Programs and Designs*. Addison-Wesley Publishing Company, Massachusetts.
- Microsoft Corporation (1997) *Microsoft Visual Basic, Professional Edition, Version 5.0*. www.microsoft.com/vbasic/.
- ObjectSpace, Inc. (1997) *ObjectSpace JGL: The Generic Collection Library for Java*. <http://www.objectspace.com/jgl/>.
- Ousterhout KK (1994) *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, Massachusetts.
- Rice, P (1996) *NewFeatures*. http://www.emblnet.org/EGCGdoc/Program_Manual/newfeatures.html.
- Robinson AJ, Flores TP (1997) Novel Techniques for Visualising Biological Information. In *Proceedings of the Fifth International Conference on Intelligent Systems in Molecular Biology* (Ed.: Gaasterland et al.). The AAAI Press, Menlo Park, CA, USA. pp241-249.
- Schilt H (1995) *C++: The Complete Reference, Second Edition*. McGraw-Hill, Inc.
- Searls DB (1995) *bioTK: Componentry for Genome Informatics Graphical User Interfaces*. *Gene* 163:1-16.
- Smith SW et al. (1994) The Genetic Data Environment: an expandable GUI for multiple sequence analysis. *CABIOS* 10(6):671-675.
- Sommerville I (1992) *Software Engineering*. Addison-Wesley Ltd.
- Stroustrup B (1997) *The C++ Programming Language (2nd Edition)*. Addison-Wesley Publishing Company, Massachusetts.

Sun Microsystems Inc (1994) The Java Language. A White Paper. <http://www.javasoft.com/>

Symantec Corporation (1996) <http://cafe.symantec.com/>.

Toldo LIG (1997) JAMBW 1.1: Java-based Molecular Biologists' WorkBench. CABIOS 13(4): 475-476.

Verplank W (1988) Graphical challenges in designing object-oriented user interfaces. In Helander M Ed. Handbook of Human-Computer Interaction. North-Holland: Elsevier Science Publishers, p365-376.

Appendix 1: Class Descriptions

Class	Description
CharacterCan	Canvas that displays a selectable character or string with a specified font, colour and background. Selection by mouse down events swaps font and background colours
CharacterRect	Optimized, lightweight version of CharacterCan
CustomizeDoF	GUI macrocomponent that allows the user to customize the appearance of features on the feature diagram
CustomizeDos	GUI macrocomponent that allows the user to customize the residue and base colours on the sequence diagram
DBDetails	Abstract class defining an object that holds information about the standard types of feature for a data bank format
DBSeqInfo	Abstract class that describes an information holder for a sequence and its annotation
DeleteFeatureDialog	Dialog GUI that asks the user to confirm a potentially destructive action
DiagramOfFeatures	Resizable canvas displaying a clickable, multilayer diagram of the features in the feature table
DiagramOfFeaturesGUI	GUI macrocomponent that holds the feature diagram and allows the user to add, edit, and delete features, as well as display more information about them and customize their appearance
DiagramOfSequence	Resizable canvas that displays the sequence in a clickable, multicolour format. The font may also be dynamically resized
DiagramOfSequenceGUI	GUI macrocomponent that holds the sequence diagram and allows the user to alter the diagram's scale, select regions, make new features based on selections
DoFListener	Interface through which DiagramOfFeatures communicates with its parent.
DoSGUIListener	Interface through which DiagramOfSequenceGUI communicates with its parent
DoSListener	Interface through which DiagramOfSequence communicates with its parent
DoSManager	Manages and integrates the activities of DiagramOfSequence and DiagramOfSequenceGUI
DoSManagerListener	Interface through which DoSManager communicates with its parent
DummyCanvas	Canvas used as a dummy component for GUI layouts
FeatureColorsROR	Read only reference that holds hashables of colours against their names
FeatureInputForm	GUI macrocomponent that allows a user to enter information about a feature
FeatureInputFormListener	Interface through which FeatureInputForm communicates with its parent
FeatureStore	Interface that defines services to access feature information
FeatureTableEntry	Abstract class that defines the attributes and public methods of a feature table entry
FeatureTableEntryPredicate	Binary predicate used as a comparator for sorting an array of feature table entries
FrameOkayCancelDialog	Pseudo-modal frame that simulates JDK Dialog and provides <code>€Okay</code> and <code>€Cancel</code> buttons.
FTEd_ResourceRegistry	A class that acts as a global notice board that can be accessed from any point in the object hierarchy.

Table continued on next page.

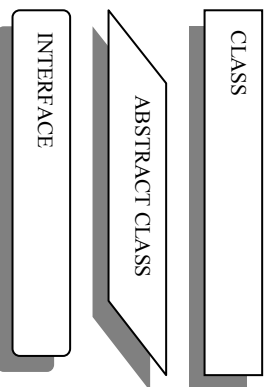
Continued from previous page.

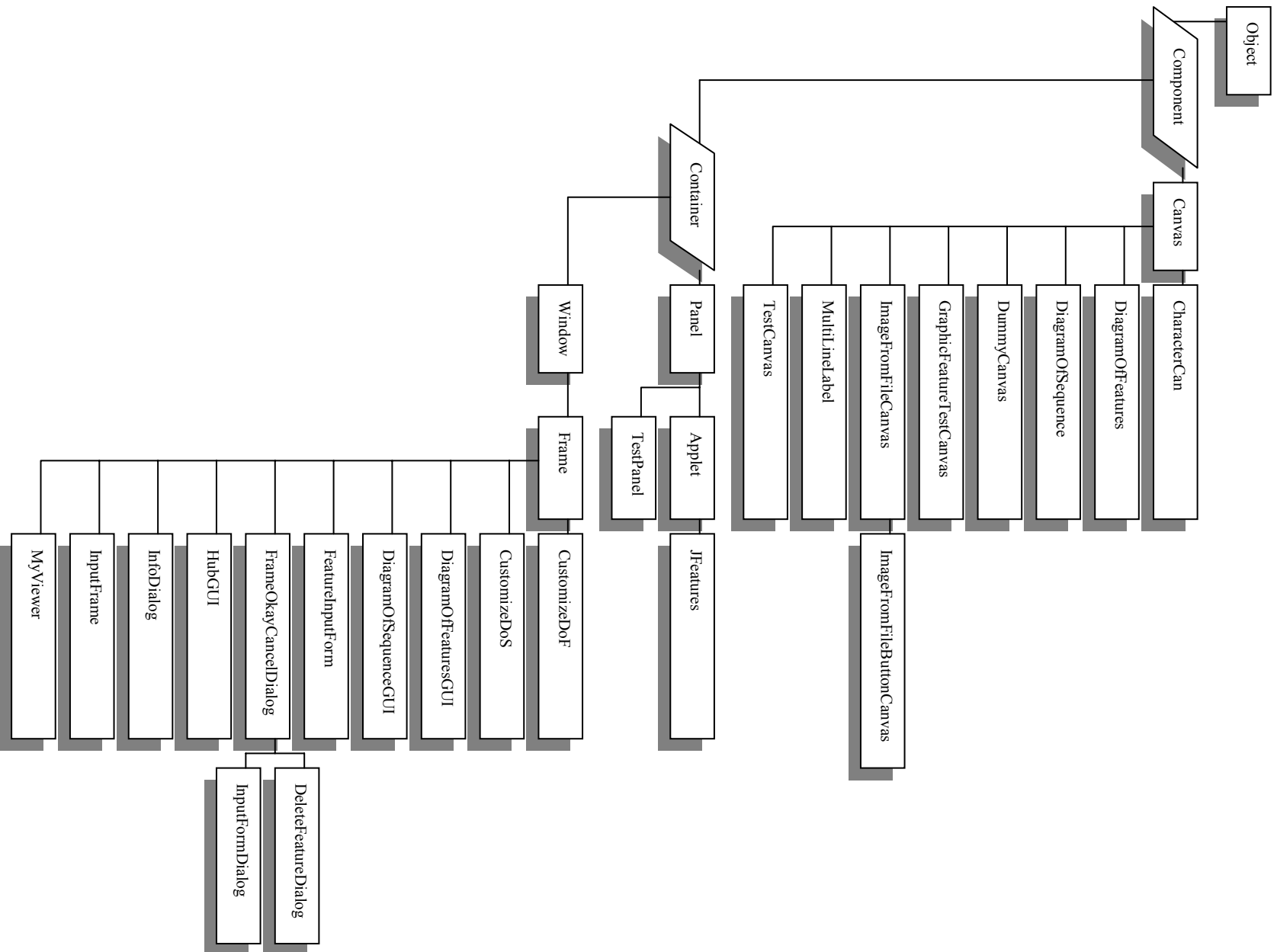
Class	Description
gblHelper	A class with static methods that facilitate use of JDK GridBagLayout and GridBagConstraints layout managers.
GraphicFeatureTestCanvas	Canvas that previews the appearance of a feature as it is being customized in CustomizedDOF.
Hub	Central object that coordinates the actions of the other modules.
HubGUI	GUI macrocomponent providing access to basic IO functions.
HubGUIListener	Interface through which HubGUI communicates with Hub
ImageFromFileButtonCanvas	Canvas that loads an image and creates a button out of a specified area of that image.
ImageFromFileCanvas	Canvas that loads an image from file
InfoDialog	Pseudo-modal information dialog box
InputFormDialog	Pseudo-modal dialog box for the FeatureInputForm
InputFormDialogListener	Interface through which the InputFormDialog communicates with the FeatureInputForm.
InputFrame	Pop-up box that provides a text-area in which the user can paste input.
JFeatures	Applet that initializes the feature table editor program when the user presses a <code>start</code> image button.
Locker	Interface that facilitates pseudo-modal behavior.
MultiInLabel	Label widget that holds multiple lines.
MyDrawnShape	Painting tool to draw a number of shapes onto a component
MyIO	Describes an object that organizes input and output.
MyIOListener	Interface through which MyIO communicates with its parent.
MyUtils	General utility class providing often used methods.
MyViewer	Pop-up read-only text box used to provide output that the user can select and copy.
ObjectFlatFileConverter	Interface through which MyIO communicates with an object that translates to and from data bank format.
SequenceColors	Information holder that provides a default colour scheme for the diagram of features.
SingleChildRestriction	An object that implements this interface may only spawn a single instance of the class that uses the interface.
SwissDetails	Holds information about standard feature types in the SwissProt data bank.
SwissTEntry	Holds information about SwissProt feature table entries
SwissInfo	Holds information about a SwissProt sequence and its annotation.
SwissProtTranslator	Describes an object that translates between data bank entries in SwissProt format SwissInfo objects.
TestCanvas	Non-resizable dummy component for GUI layouts.
TestPanel	Non-resizable dummy panel for GUI layouts.
Triggerable	Interface implemented by objects that do something significant as the result of a single action in another object.
TwoWayHashable	A hashable in which the key for a given property can be looked up, as well as the property for a given key.
TypeGProperties	An object that holds information about the graphical properties of a feature type in the feature diagram.

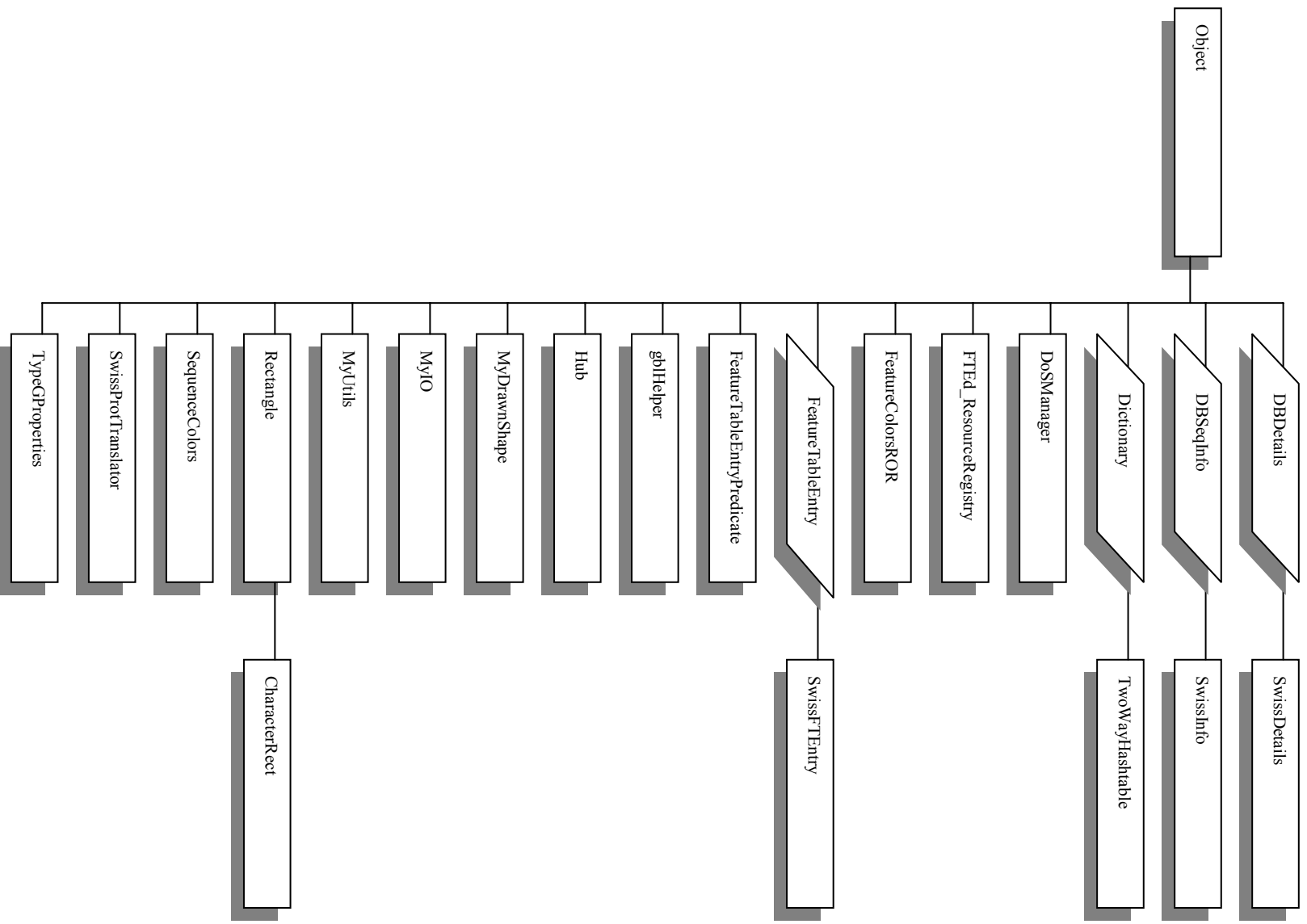
Appendix 2: Class Hierarchy

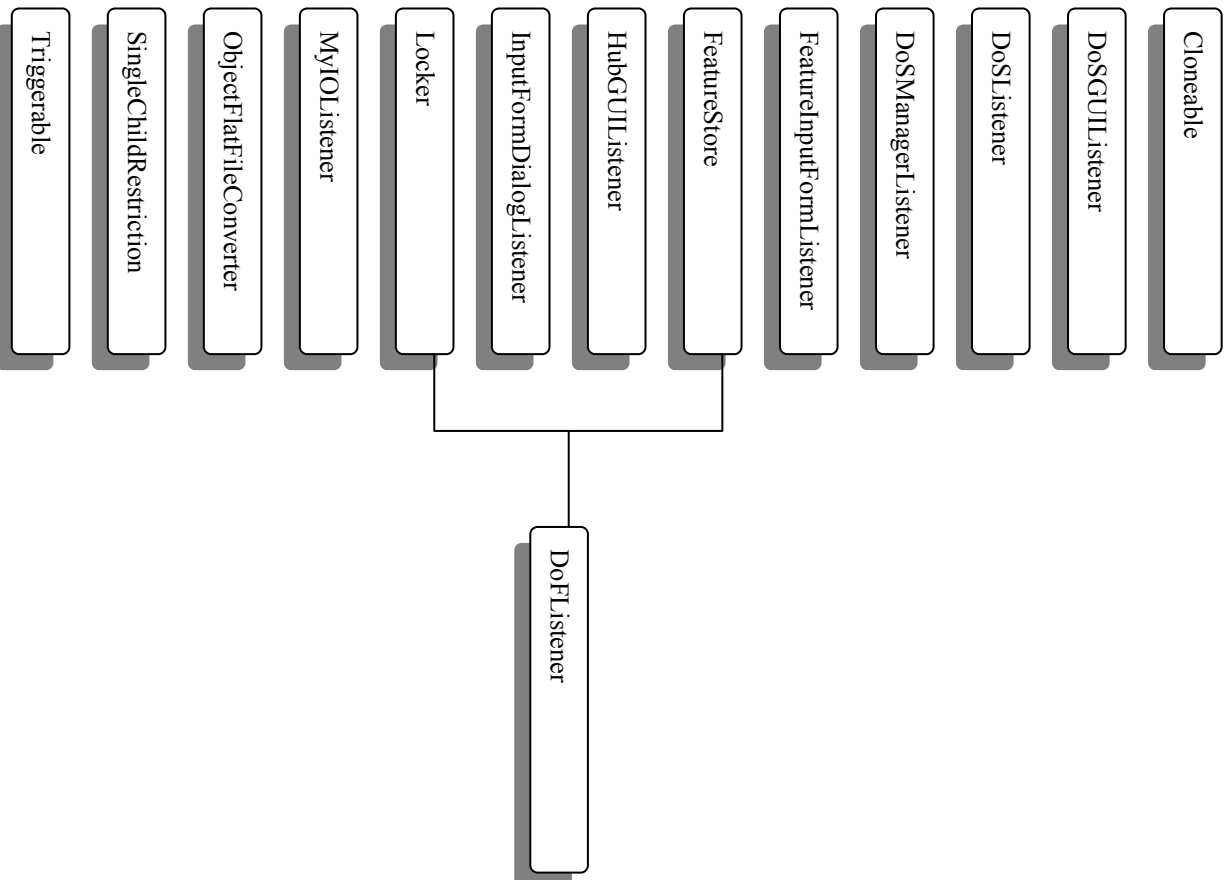
The following pages contain diagrams that show how the classes used in J-Features are related to each other and to those in the Java Development Kit (JDK) provided with Java 1.02. The diagrams use the scheme of Flanagan (1996) to represent the different types of class, given below. A line connecting a class X, on the left, to a class Y, on the right indicates that Y is a subclass of X.

Key:









Appendix 3: J-Features Source Code

The following pages contain the source code to the J-Features applet. The classes and interfaces are given in alphabetical order in one continuous section. This code is written in the Java language, version 1.02 (Sun Microsystems, 1994).

```

/*
 * CLASS: CharacterCan
 *
 * This class encapsulates a box component containing a character or string.
 * Mouse-down events toggle selection and cause the colours of the font and
 * background to be reversed. Objects of this class were originally used to
 * represent residues or bases in the DiagramOfSequence class but, as components,
 * required heavy computational overhead when their numbers were great. CharacterCan
 * has been replaced by CharacterRect everywhere except in the CustomizeDoS
 * macrocomponent.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

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

```

public class CharacterCan extends Canvas
{
    Font f;
    int fontsize = 38;
    String fontname = "Courier";
    FontMetrics fm;
    String ch;
    Color fg, bg;
    int pos;
    boolean selected = false;
    boolean selectable = true;

    public CharacterCan ()
    {
        ch = "A"; //Dummy

        fg = Color.white;
        bg = Color.black;

        this.setBackground(bg);
    }

    public void paint(Graphics g)
    {
        if(selected) {g.setColor(bg);}

        else {g.setColor(fg);}
    }
}

```

```

        g.setFont(f);

        fm = g.getFontMetrics(f);

        g.drawString(ch,0,fm.getAscent());
    }

    public Dimension preferredSize()
    {
        Graphics g = this.getGraphics();
        fm = g.getFontMetrics(f);
        return new Dimension(fm.charWidth(ch.charAt(0)),
fm.getAscent()+fm.getDescent());
    }

    public Dimension minimumSize()
    {
        return preferredSize();
    }

    public boolean mouseDown(Event e, int x, int y)
    {
        if( selectable == false) {return true;}

        if( selected ) {deselect();}

        else {select();}

        return true;
    }

    //Public interface methods

    // selection / deselection
    // selection swaps the background and foreground colours

    public void setSelectable(boolean b)
    {
        selectable = b;
    }
}

```

```

public boolean isSelectable()
{
    return selectable;
}

public boolean isSelected()
{
    return selected;
}

public void select()
{
    selected = true;
    this.setBackground(fg);
}

public void deselect()
{
    selected = false;
    this.setBackground(bg);
}

public void select(boolean paintMe)
{
    select();
    if(paintMe){ this.repaint(); }
}

public void deselect(boolean paintMe)
{
    deselect();
    if(paintMe){ this.repaint(); }
}

// changing the size

public void bigger(int increment)
{
    fontsize += increment;
    refontify();
}

public void smaller(int decrement)
{
    fontsize -= decrement;
    refontify();
}

public void setFontSize(int size)
{
    fontsize = size;
    refontify();
}

protected void refontify()
{
    f = new Font(fontname, Font.PLAIN, fontsize);
}

//setting the character or string

public void setChar(String ch)
{
    this.ch = ch;
}

public void setChar(char c)
{
    this.ch = String.valueOf(c);
}

//setting the color

public void setBackgroundColor(Color b)
{
    bg = b;
    this.setBackground(bg);
}

public void setForegroundColor(Color f)
{
    fg = f;
}
}

```

```

/*
 * CLASS: CharacterRect
 *
 * This class encapsulates a rectangle that contains a character or string.
 * CharacterRect was written as a light-weight alternative to CharacterCan
 * that could be easily painted onto a canvas and be used to simulate
 * component-like behavior. Objects of this class are used in the
 * DiagramOfSequence class to represent the residues or bases of a
 * sequence.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.awt.*;

public class CharacterRect extends Rectangle
{
    Canvas myParent;
    String ch;
    Color myColor, background;
    int ascent;
    boolean selected;

    public CharacterRect (Canvas myP, String c, Color myC, Color bg)
    {
        myParent = myP;
        ch = c;
        myColor = myC;
        background = bg;
        selected = false;
    }

    public void paint(Graphics g)
    {
        if(selected)
        {
            g.setColor(myColor);
            g.fillRect(x,y,width,height);
            g.setColor(background);
            g.drawString(ch, x, y+ascent);
        }
    }
}

else
{
    g.setColor(background);
    g.fillRect(x,y,width, height);
    g.setColor(myColor);
    g.drawString(ch, x, y+ascent);
}

public void paint()
{
    paint(myParent.getGraphics());
}

public void setAscent(int a)
{
    ascent = a;
}

public void select()
{
    selected = true;
}

public void deselect()
{
    selected = false;
}

public boolean isSelected()
{
    return selected;
}
}

/*
 * CLASS: CustomizeDoF
 *
 * This class encapsulates a GUI frame containing sample features,
 * choices and buttons that allow the graphical properties of the
 * features in the feature table to be changed.
 * @author William Valdar

```

```

* @version 1.0 (September 1997)
*
*/

```

```

import java.awt.*;
import gjt.*;
import java.util.*;

```

```

public class CustomizeDoF extends Frame
{

```

```

    DiagramOfFeatures dof;
    GraphicFeatureTestCanvas fcbig;
    GraphicFeatureTestCanvas fcsmall;

```

```

    FTED_ResourceRegistry rr;
    FeatureColorsROR featureColorsROR;

```

```

    TypeGProperties tgp;
    TypeGProperties prev_tgp;
    String type;

```

```

    Choice ch_type;
    Choice ch_color;
    Choice ch_shape;
    Choice ch_height;
    Checkbox cb_show;

```

```

    Button bu_revert;
    Button bu_apply;
    Button bu_close;

```

```

    GridBagLayout gridbag;
    gblHelper gbl;

```

```

    public CustomizeDoF(DiagramOfFeatures dof)
    {
        super("Customize Feature Diagram");

        this.dof = dof;

```

```

        rr = FTED_ResourceRegistry.instance();
        featureColorsROR = rr.getFeatureColorsROR();

```

```

        instantiateComponents();
        setUpChoices();
        setUpCanvases();
        layoutComponents();

```

```

        this.pack();
        this.setResizable(false);
        this.show();
    }

```

```

    private void instantiateComponents()
    {

```

```

        ch_type = new Choice();
        ch_color = new Choice();
        ch_shape = new Choice();
        ch_height = new Choice();

```

```

        cb_show = new Checkbox("Show");

```

```

        bu_revert = new Button("Revert");
        bu_apply = new Button("Apply");
        bu_close = new Button("Close");
    }

```

```

    private void layoutComponents()
    {

```

```

        gridbag = new GridBagLayout();
        gbl = new gblHelper();

```

```

        Panel chp = layoutChoicePanel();
        Panel bp = layoutButtonPanel();
        Panel cp = layoutCanvasPanel();

```

```

        Panel contain_all = new Panel();
        contain_all.setLayout(gridbag);

```

```

        Insets myinsets = new Insets(0,0,0,0);

```

```

        gbl.constrain(contain_all, cp, 0,0, 1,1, GridBagConstraints.NONE,
GridBagConstraints.NORTHWEST,
                0.0, 0.0, myinsets);
        gbl.constrain(contain_all, chp, 1,0, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.NORTHWEST,
                0.0, 0.0, myinsets);
        gbl.constrain(contain_all, bp, 0,1, 2,1, GridBagConstraints.BOTH,
GridBagConstraints.CENTER,
                0.0, 0.0, myinsets);

        Box b1 = new Box(contain_all, "Customize Feature Graphics");

        this.add("Center", b1);
        this.pack();
        this.setResizable(false);
    }

    private Panel layoutChoicePanel()
    {
        Panel ap = new Panel();
        ap.setLayout(new GridLayout(4,2,10,5));

        ap.add(new Label("Colour", Label.CENTER));
        ap.add(new Label("Shape", Label.CENTER));
        ap.add(ch_color);
        ap.add(ch_shape);
        ap.add(new Label("Height", Label.CENTER));
        ap.add(new DummyCanvas(0,0)); //lightweight dummy
        ap.add(ch_height);
        ap.add(cb_show);

        Box b1 = new Box(ap, "Appearance");

        Panel chp = new Panel();
        chp.setLayout(gridbag);

        Box b2 = new Box(ch_type, "Type Of Feature");

        Insets myinsets = new Insets(0,0,0,0);

        gbl.constrain(chp, b2, 0,0, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.WEST,
                0.0, 0.0, myinsets);

```

```

        gbl.constrain(chp, b1, 0,1, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.WEST,
                0.0, 0.0, myinsets);
        gbl.constrain(chp, bu_revert, 0,2, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.WEST,
                0.0, 0.0, myinsets);

        return chp;
    }

    public Panel layoutButtonPanel()
    {
        Panel bp = new Panel();
        bp.setLayout(new GridLayout(1,3));

        bp.add(bu_apply);
        bp.add(new DummyCanvas(0,0)); //lightweight dummy
        bp.add(bu_close);

        Panel bp1 = new Panel();
        bp1.setLayout(new GridLayout(2,1));

        Separator sep = new Separator();
        bp1.add(sep);
        bp1.add(bp);

        return bp1;
    }

    public Panel layoutCanvasPanel()
    {
        Panel cp = new Panel();
        cp.setLayout(new ColumnLayout());

        cp.add(new EtchedBorder(fcbig));
        cp.add(new Label("(Small Features)", Label.CENTER));
        cp.add(new EtchedBorder(fsmall));

        return cp;
    }

    private void setUpChoices()
    {

```

```

//choice of types

Vector tempv = MyUtils.getVectorOfEnumeration( rr.getAllTypes() );
MyUtils.sortVectorOfStringsFwd( tempv );
MyUtils.choiceItemsFromStringVector( ch_type, tempv );
tempv = null;

//choice of colours

MyUtils.choiceItemsFromStringVector( ch_color,
featureColorsROR.getColorNames() );

//choice of shape

ch_shape.addItem("rectangle");
ch_shape.addItem("ellipse");

//choice of height

ch_height.addItem("low");
ch_height.addItem("medium");
ch_height.addItem("high");
}

private void setUpCanvases()
{
//Setup the canvases

type = ch_type.getSelectedItem();
prev_tgp = rr.getTypeGProperties( type );
tgp = (TypeGProperties) prev_tgp.clone();
fcbig = new GraphicFeatureTestCanvas(tgp,
GraphicFeatureTestCanvas.NOEGGS);
fcsmall = new GraphicFeatureTestCanvas(tgp,
GraphicFeatureTestCanvas.EGGSONLY);

resetChoices();
}

private void refreshTestCanvases()
{
fcbig.setTypeGProperties(tgp);
fcsmall.setTypeGProperties(tgp);
}

```

```

}

private void resetChoices()
{
ch_color.select( (String)
featureColorsROR.getAvailableColorHash().getKeyFor( tgp.fill_color ) );
switch(tgp.height)
{
case TypeGProperties.LOW:
ch_height.select("low");
break;
case TypeGProperties.MEDIUM:
ch_height.select("medium");
break;
case TypeGProperties.HIGH:
ch_height.select("high");
break;
}
}
switch(tgp.shape)
{
case MyDrawnShape.RECTANGLE:
ch_shape.select("rectangle");
break;
case MyDrawnShape.OVAL:
ch_shape.select("oval");
break;
}
cb_show.setState( tgp.visible );
}

public boolean action(Event e, Object arg)
{
if(e.target == ch_type)
{
changeType(arg);
return true;
}
if(e.target == ch_color)
{
changeColor(arg);
return true;
}
if(e.target == ch_shape)

```

```

    {
        changeShape(arg);
        return true;
    }
    if(e.target == ch_height)
    {
        changeHeight(arg);
        return true;
    }
    if(e.target == cb_show)
    {
        changeShow();
        return true;
    }
    if(e.target == bu_apply)
    {
        apply();
        return true;
    }
    if(e.target == bu_revert)
    {
        revert();
        return true;
    }
    if(e.target == bu_close)
    {
        close();
        return true;
    }
    return false;
}

private void changeType(Object arg)
{
    type = (String) arg;
    prev_tgp = (TypeGProperties) rr.getTypeGProperties(type);
    tgp = (TypeGProperties) prev_tgp.clone();
    refreshTestCanvases();
    resetChoices();
}

private void changeColor(Object arg)
{
    tgp.fill_color = (Color) featureColorsROR.getAvailableColorHash().get(arg);
    refreshTestCanvases();
}

private void changeShape(Object arg)
{
    if(arg.equals("rectangle"))
    {
        tgp.shape = MyDrawnShape.RECTANGLE;
    }
    if(arg.equals("ellipse"))
    {
        tgp.shape = MyDrawnShape.OVAL;
    }
    refreshTestCanvases();
}

private void changeHeight(Object arg)
{
    if(arg.equals("low"))
    {
        tgp.height = TypeGProperties.LOW;
    }
    else if(arg.equals("medium"))
    {
        tgp.height = TypeGProperties.MEDIUM;
    }
    else if(arg.equals("high"))
    {
        tgp.height = TypeGProperties.HIGH;
    }
    refreshTestCanvases();
}

private void changeShow()
{
    tgp.visible = cb_show.getState();
    refreshTestCanvases();
}

private void apply()
{
    rr.replaceTypeGProperties(type, (TypeGProperties) tgp.clone());
}

```

```

        dof.repaint();
    }

    private void revert()
    {
        tgp = prev_tgp;
        refreshTestCanvases();
        resetChoices();
    }

    private void close()
    {
        die();
    }

    public void die()
    {
        this.hide();
        this.dispose();
    }
}

```

```

/*
 * CLASS: CustomizeDoS
 *
 * This class encapsulates a GUI frame that contains a sample
 * residue/base, choices and buttons that allow the user to
 * customize the appearance of the characters in the
 * DiagramOfSequence.
 * @author William Valdar
 * @version 1.0 (September 1997)
 *
 */

```

```

import java.awt.*;
import gjt.*;

```

```

public class CustomizeDoS extends Frame
{
    DoSGUIListener listener;
    CharacterCan cc;

    Choice ch_dnaprot;
    Choice ch_res;
    Choice ch_base;
    Choice ch_color;

    Button bu_apply;
    Button bu_reset;
    Button bu_close;

    GridBagLayout gridbag;
    gblHelper gbl;

    SequenceColors sequenceColors;

    public CustomizeDoS(DoSGUIListener l)
    {
        super("Customize Sequence Colours");
        sequenceColors = FTED_ResourceRegistry.instance().getSequenceColors();

        listener = l;

        instantiateComponents();
        setUpChoices();
        layoutComponents();
        this.pack();
        this.setResizable(false);
        this.show();
    }

    public void instantiateComponents()
    {
        cc = new CharacterCan();

        ch_dnaprot = new Choice();
        ch_res = new Choice();
        ch_base = new Choice();
        ch_color = new Choice();
    }
}

```

```

        bu_apply = new Button("Apply");
        bu_reset = new Button("Reset All");
        bu_close = new Button("Close");
    }

    private void layoutComponents()
    {
        gridbag = new GridBagLayout();
        gbl = new gblHelper();

        Panel ccp = layoutCC();
        Panel chp = layoutChoices();
        Panel bp = layoutButtons();

        Panel contain_all = new Panel();
        contain_all.setLayout(gridbag);

        Insets myinsets = new Insets(0,0,0,0);

        Separator sep = new Separator();

        gbl.constrain(contain_all, ccp, 0,0, 1,1, GridBagConstraints.BOTH,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);
        gbl.constrain(contain_all, chp, 1,0, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);
        gbl.constrain(contain_all, bp, 0,1, 2,1, GridBagConstraints.BOTH,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);

        Box title_box = new Box(contain_all, "Customize Sequence Colours");

        this.add("Center", title_box);
    }

    private Panel layoutCC()
    {
        Panel p1 = new TestPanel(150,0);
        p1.setBackground(Color.black);
        p1.setLayout(gridbag);

        gbl.constrain(p1, cc, 0,0, 1,1, GridBagConstraints.NONE,
GridBagConstraints.CENTER,
                    0.0, 0.0, 0,0,0,0);

        return new Box(p1, "Preview");
    }

    private Panel layoutChoices()
    {
        Box b1 = new Box(ch_dnaprot, "Dna / Protein");

        Panel p2 = new Panel();
        p2.setLayout(new GridLayout(1,2, 20,0));
        p2.add(ch_res);
        p2.add(ch_base);

        Box b2 = new Box(p2, "Residue / Base");

        Box b3 = new Box(ch_color, "Colour");

        Panel chp = new Panel();
        chp.setLayout(new GridLayout(3,1));
        chp.add(b1);
        chp.add(b2);
        chp.add(b3);

        return chp;
    }

    private Panel layoutButtons()
    {
        Panel bp = new Panel();
        bp.setLayout(new GridLayout(1,3,10,0));

        bp.add(bu_apply);
        bp.add(bu_reset);
        bp.add(bu_close);

        Panel bp1 = new Panel();
        bp1.setLayout(new GridLayout(2,1,0,0));

```

```

        bp1.add(new Separator());
        bp1.add(bp);

        return bp1;
    }

    private void setUpChoices()
    {
        ch_dnaprot.addItem("Dna");
        ch_dnaprot.addItem("Protein");

        MyUtils.choiceItemsFromStringVector(ch_res,
sequenceColors.getAvailableResidues());

        MyUtils.choiceItemsFromStringVector(ch_base,
sequenceColors.getAvailableBases());

        MyUtils.choiceItemsFromStringVector(ch_color,
sequenceColors.getAvailableColorNames());

        ch_dnaprot.select("Dna");
        ch_res.disable();
        ch_base.enable();

        cc.setFontSize(72);
        cc.setBackgroundColor( Color.black );
        refreshCC();
    }

    public boolean action(Event e, Object arg)
    {
        if(e.target == ch_dnaprot)
        {
            changeDnaprot(arg);
            return true;
        }
        if(e.target == ch_res)
        {
            changeRes(arg);
            return true;
        }
        if(e.target == ch_base)
        {

```

```

            changeBase(arg);
            return true;
        }
        if(e.target == ch_color)
        {
            changeColor(arg);
            return true;
        }
        if(e.target == bu_reset)
        {
            reset();
            return true;
        }
        if(e.target == bu_apply)
        {
            apply();
            return true;
        }
        if(e.target == bu_close)
        {
            close();
            return true;
        }
        return false;
    }

    private void changeDnaprot(Object arg)
    {
        if(arg.equals("Dna"))
        {
            ch_res.disable();
            ch_base.enable();
        }
        else if(arg.equals("Protein"))
        {
            ch_base.disable();
            ch_res.enable();
        }
        refreshCC();
    }

    private void changeRes(Object arg)
    {

```

```

        refreshCC();
    }

    private void changeBase(Object arg)
    {
        refreshCC();
    }

    private void changeColor(Object arg)
    {
        cc.setForegroundColor( (Color) sequenceColors.getAvailableColors().get(arg) );
        cc.repaint();
    }

    private void refreshCC()
    {
        String temps = null;

        Choice charsource = null;

        if(ch_res.isEnabled())
        {
            String s = ch_res.getSelectedItem();
            if(s.length() > 1)
            {
                cc.setChar('X');
            }
            else
            {
                cc.setChar(s);
            }
            Color c = (Color) sequenceColors.getResidueColor(s);
            cc.setForegroundColor( c );
            ch_color.select( (String)
sequenceColors.getAvailableColors().getKeyFor(c) );
        }
        if(ch_base.isEnabled())
        {
            String s = ch_base.getSelectedItem();
            if(s.length() > 1)
            {
                cc.setChar('X');
            }
            else
            {
                cc.setChar(s);
            }
            Color c = (Color) sequenceColors.getBaseColor(s);
            cc.setForegroundColor( c );
            ch_color.select( (String)
sequenceColors.getAvailableColors().getKeyFor(c) );
        }
        cc.deselect(true);
    }

    private void reset()
    {
        FTED_ResourceRegistry.instance().setSequenceColors( new SequenceColors());
        sequenceColors = FTED_ResourceRegistry.instance().getSequenceColors();
        listener.applyColorChange();
    }

    private void apply()
    {
        String ch = null; Color c = null;

        c = (Color) sequenceColors.getAvailableColors().get(
ch_color.getSelectedItem() );

        if(ch_res.isEnabled())
        {
            ch = ch_res.getSelectedItem();
            sequenceColors.replaceResidueColor(ch,c);
        }
        if(ch_base.isEnabled())
        {
            ch = ch_base.getSelectedItem();
            sequenceColors.replaceBaseColor(ch,c);
        }
        listener.applyColorChange();
    }

    private void close()
    {
        die();
    }

```

```

    }

    public void die()
    {
        this.hide();
        this.dispose();
    }

}

/*
 * CLASS: DBDetails
 *
 * This abstract class defines an object that holds information
 * about the standard features keys (or types) for a database.
 * Subclasses of this have database-dependent implementations
 * of the method it defines. This may not be the most effective
 * or appropriate way to store additional information about a
 * particular database.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.util.*;

public abstract class DBDetails
{
    public Vector getStdTypes(){return null;};
    public Hashtable getTypesGProps(){return null;};
}

/**
 * CLASS: DBSeqInfo
 *
 * DBSeqInfo is an abstract class defining an object that holds
 * information about a database entry. Most of the JFeatures program
 * considers the subject database entry as an object of this type.
 * The actual object must therefore be an instance of a subclass of

```

```

 * DBSeqInfo. This class was made abstract to allow for database
 * format - dependent implementations of its methods. See my subclass,
 * SwissInfo.
 *
 * It may have been more useful to implement some of the 'getter' and
 * 'setter' methods in this abstract class, since it is unlikely
 * subclasses will implement these simple methods differently.
 * @author William Valdar
 * @version 1.0
 */

```

```

import java.util.*;

public abstract class DBSeqInfo extends Object implements Cloneable
{
    public abstract String getFormat();
    public abstract String getAccession();
    public abstract String getId();
    public abstract String getSequence();
    public abstract Vector getFeatures();
    public abstract Vector getOriginalInfo();

    public abstract void setAccession(String accession);
    public abstract void setId(String id);
    public abstract void setSequence(String sequence);
    public abstract void setFeatures(Vector features);
    public abstract void setOriginalInfo(Vector original_info);

    public abstract boolean hasSequence();
    public abstract boolean hasFeatures();
    public abstract boolean hasOriginalInfo();
    public abstract Object clone();
}

```

```

/*
 * CLASS: DeleteFeatureDialog
 *
 * This class extends the FrameOkayCancelDialog and encapsulates
 * a dialog box that has the power to sanction the deletion of a
 * particular feature. On construction an instance of this class
 * takes the index of the feature, and when its okay() method is

```

```

* invoked it requests the listener to delete the feature at
* that index.
* @author William Valdar
* @version 1.0 (September 1997)
*/

import java.awt.*;
import javax.swing.*;

public class DeleteFeatureDialog extends FrameOkayCancelDialog
{
    DoFListener listener;

    int index;

    public DeleteFeatureDialog(Locker locker, boolean modal, String subtitle, String message,
DoFListener l, int index)
    {
        super(locker, modal, subtitle, message);

        listener = l;
        this.index = index;

        this.show();
    }

    public void okay()
    {
        listener.delete(index);
    }
}

/*
* CLASS: DiagramOfFeatures
*
* This class encapsulates a diagram that can dynamically represent
* the information in a feature table. It is essentially a canvas on
* which is painted shapes of various size and colour, representing
* features, along a central horizontal line, representing the
* sequence. The appearance of a feature depends on the information

```

```

* in the TypeGProperties object associated with the name of that
* feature's key (type) stored in the FTED_ResourceRegistry. Every
* time the paint method is called the dimensions of the diagram are
* recalculated relative to the size of the canvas. When features are
* selected they become brighter and move to front. The double
* for-loop in the mouseDown() implementation allows the user to
* access any feature no matter how many other features overlap or
* cover it. Selections and deselections do not invoke the paint
* method, causing only local repaints.
*
* The DiagramOfFeatures does not require a handle on actual data
* and should be passed messages and updated information whenever
* the data is changed. The class has been optimized for speed.
* Double buffering could improve its performance further.
*
* @author William Valdar
* @version 1.0 (September 1997)
*/

```

```

import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.applet.*;

public class DiagramOfFeatures extends Canvas
{
    DoFListener listener;

    private int num_features = 0;

    private FeatureTableEntry[] features;
    private MyDrawnShape[] features_mirror;
    private String[] types_mirror;

    protected FTED_ResourceRegistry rr;

    private Rectangle mybounds;
    private int ox,oy,maxw,maxh;
    private double unit;
    private MyDrawnShape lastrect;
    private int lasti = 0;
}

```

```

protected int borderwidth = 5;
protected int seq_length = 500;
protected int smallestClickableWidth = 5;
public static final int eggtimerW = 20;
public static final int eggtimerH = TypeGProperties.HIGH;
protected Color background_color = Color.white;
protected Color sequence_color = Color.black;

protected int line_thickness = 1;
protected int feature_height = 30;
protected int index_to_front = 0;

public DiagramOfFeatures(DoFListener l)
{
    listener = l;

    this.setBackground(background_color);

    rr = FTED_ResourceRegistry.instance();
}

public void paint(Graphics g)
{
    refreshScalers();
    refreshMirror();

    lastrect = null;

    //Sequence line
    g.setColor(sequence_color);
    g.fillRect(ox,oy-3,maxw, 6);

    if(features_mirror != null)
    {
        MyDrawnShape drect;

        for(int i=0; i<num_features; i++)
        {
            drect = features_mirror[i];
            drect.paint();
        }
    }
}

```

```

    }
}

public void clearAll()
{
    features_mirror = null;
    types_mirror = null;
    num_features = 0;
    repaint();
}

public void setInfo(FEATURETABLEENTRY[] features, int seq_length)
{
    this.seq_length = seq_length;
    this.features = features;
    num_features = features.length;
    lastrect = null;
    lasti = 0;
    refreshMirror();
    repaint();
}

public int getLastIndex()
{
    return lasti;
}

public Dimension getPreferredSize()
{
    return preferredSize();
}

public Dimension preferredSize()
{
    return new Dimension(300, 100);
}

protected void refreshScalers()
{
    mybounds = this.bounds();
    ox = mybounds.x + borderwidth;
    oy = mybounds.height/2;
    maxw = mybounds.width-(2*ox);
}

```

```

        maxh = mybounds.height/2-(mybounds.y+borderwidth);
        unit = (double)maxw/(double)seq_length;
    }

protected void refreshMirror()
{
    if(features != null)
    {
        features_mirror = new MyDrawnShape[num_features];
        types_mirror = new String[num_features];

        FeatureTableEntry fte; Point loc; String type;
        MyDrawnShape temprect; String tempstring;
        TypeGProperties tgp;

        for(int i =0; i<num_features; i++)
        {
            fte = features[i];
            loc = fte.getLocation();

            int start = loc.x;
            int end = loc.y;
            type = fte.getType();

            types_mirror[i] = new String( type + " (" +start+"-"+
"+end+)");

            if(rr.hasTypeGPropertiesFor(type) == false)
            {
                rr.addTypeGProperties(type, new
TypeGProperties());
            }

            tgp = rr.getTypeGProperties(type);

            int w = (int)(unit*(double)(end-start));

            if(w < smallestClickableWidth)
            {
                int x = ox+(int)(unit * (double) start) + (w-
                eggtimerW)/2;

                int y = oy-eggtimerH;
                int h = 2*eggtimerH;

                temprect = new MyDrawnShape(this,
                line_thickness,
                MyDrawnShape.EGGTIMER,
                x,y,eggtimerW, h,
                tgp.visible);
            }
            else
            {
                int x = ox+(int)(unit * (double) start);
                int y = oy-tgp.height;
                int h = 2*tgp.height;

                temprect = new MyDrawnShape(this,
                line_thickness,
                tgp.shape,
                x,y,w,h,
                tgp.visible);
            }
            temprect.setFillColor(tgp.fill_color.darker());

            temprect.setLineColor(tgp.line_color);
            features_mirror[i] = temprect;
        }
    }
    return;
}
else
{
    num_features = 0;
    features_mirror = null;
    types_mirror = null;
    return;
}
}

```

```

    }
}

private void quickShowCurrentFeature()
{
    listener.quickShowFeature( lasti );
}

private boolean userHasSelected(MyDrawnShape direct, int x, int y)
{
    return direct.isVisible() && direct.inside(x,y);
}

public boolean mouseDown(Event e, int x, int y)
{
    //MyDrawnShape direct;

    boolean found = false; int foundi = -1;

    for(int i = (lasti + 1) ; i<num_features; i++)
    {
        if( userHasSelected( features_mirror[i], x, y ) )
        {
            found = true;
            foundi = i;
            break;
        }
    }

    if(found == false)
    {
        for(int i = 0; i < (lasti + 1); i++)
        {
            if( userHasSelected( features_mirror[i], x, y ) )
            {
                found = true;
                foundi = i;
                break;
            }
        }
    }

    if(found && foundi >= 0)

```

```

{
    if(lastrect != null)
    {
        lastrect.paint();
    }

    lastrect = features_mirror[foundi];
    lasti = foundi;
    Color fcolor = lastrect.getFillColor();
    lastrect.setFillColor(fcolor.brighter());
    lastrect.paint();

    //Set back to original color

    lastrect.setFillColor(fcolor);
    quickShowCurrentFeature();
    return true;
}
return false;
}

/*
for(int i = 0; i<num_features; i++)
{
    direct = features_mirror[i];
    if( direct != lastrect && direct.isVisible())
    {
        if( direct.inside(x,y) )
        {
            if(lastrect != null)
            {
                lastrect.paint();
            }
            lastrect = direct;
            lasti = i;
            Color fcolor = direct.getFillColor();
            direct.setFillColor(fcolor.brighter());
            direct.paint();

            //Set back to original color

            direct.setFillColor(fcolor);

```

```

        quickShowCurrentFeature();
        return true;
    }
}
return false;
}
}
*/
}

/*
 * CLASS: DiagramOfFeaturesGUI
 *
 * This class encapsulates a GUI macrocomponent that contains
 * a number of buttons and a text display, and frames the
 * DiagramOfFeatures object. The buttons provide an interface
 * to some of the editing methods and the text window
 * dynamically displays information about the last feature
 * selected on the DiagramOfFeatures.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.awt.*;
import gjt.*;

public class DiagramOfFeaturesGUI extends Frame
{
    DiagramOfFeatures dof;
    DoFListener listener;
    FeatureStore fs;

    Button bu_dispInfo;
    Button bu_add;
    Button bu_edit;
    Button bu_delete;
    Button bu_customize;
    TextField ta_currFeat;

```

```

GridBagLayout gridbag;
gblHelper gbl;

public DiagramOfFeaturesGUI(DiagramOfFeatures dof, DoFListener l)
{
    super("Feature Diagram");

    this.dof = dof;
    listener = l;

    bu_dispInfo = new Button("Display Info");
    bu_add = new Button("Add");
    bu_edit = new Button("Edit");
    bu_delete = new Button("Delete");
    bu_customize = new Button("Customize");

    ta_currFeat = new TextField(20);

    initializeComponents();
    layoutComponents();
}

private void initializeComponents()
{
    ta_currFeat.setEditable(false);
}

private void layoutComponents()
{
    gridbag = new GridBagLayout();
    gbl = new gblHelper();

    Panel control_panel = layoutControlPanel();

    Border bdof = new EtchedBorder(dof);

    Panel contain_all = new Panel();

    contain_all.setLayout(new BorderLayout());

    contain_all.add("Center", bdof);
    contain_all.add("South", control_panel);
}

```

```

        Box title_box = new Box(contain_all, "Feature Diagram");

        this.add("Center", title_box);

        this.pack();
        this.show();
    }

    private Panel layoutButtonPanel()
    {
        Panel p1 = new Panel();
        p1.setLayout(new GridLayout(1,3));
        p1.add(bu_dispInfo);
        p1.add(bu_add);
        p1.add(bu_edit);
        p1.add(bu_delete);

        Box b1 = new Box(p1, "Current Feature Menu");

        return b1;
    }

    private Panel layoutControlPanel()
    {
        Panel bp = layoutButtonPanel();
        Panel bt = new Box(ta_currFeat, "Current Feature");
        Panel bc = new Box(bu_customize, "Diagram");

        Panel p1 = new Panel();
        p1.setLayout(gridbag);

        Insets myinsets = new Insets(2,5,2,5);

        gbl.constrain(p1, bt, 0,0, 1,1, GridBagConstraints.VERTICAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);
        gbl.constrain(p1, bp, 1,0, 1,1, GridBagConstraints.VERTICAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);
        gbl.constrain(p1, bc, 2,0, 1,1, GridBagConstraints.VERTICAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);
    }

```

```

        return p1;
    }

    public void setCurrentFeatureBox(String s)
    {
        ta_currFeat.setText(s);
    }

    private void delete(int index)
    {
        if(listener.hasFeatureAt(index))
        {
            FeatureTableEntry fte = listener.getFeatureAt(index);
            String s = "Delete '" + fte.getType() + " (" + fte.getLocation().x + "-"
                + fte.getLocation().y + ")' ?";
            new DeleteFeatureDialog(listener, true, "Delete Feature", s, listener,
index);
        }
    }

    public boolean action(Event e, Object arg)
    {
        if(e.target == bu_dispInfo)
        {
            listener.showFeature(dof.getLastIndex());
            return true;
        }
        if(e.target == bu_add)
        {
            listener.newFeature();
            return true;
        }
        if(e.target == bu_edit)
        {
            listener.edit(dof.getLastIndex());
            return true;
        }
        if(e.target == bu_delete)
        {
            delete(dof.getLastIndex());
            return true;
        }
    }

```

```

        }
        if(e.target == bu_customize)
        {
            new CustomizeDoF(dof);
            return true;
        }
        return false;
    }

    public void die()
    {
        this.hide();
        this.dispose();
    }
}

```

```

/*
 * CLASS: DiagramOfSequence
 *
 * This class extends Canvas and encapsulates a diagram of a
 * sequence. Residues/bases in the diagram are represented by
 * CharacterRect objects and are manipulated by the class to
 * simulate the behavior of selectable CharacterCan objects.
 * Characters are coloured according to a customizable scheme
 * accessed through the FTED_ResourceRegistry.
 *
 * The DiagramOfSequence is intended to be scrolled. However,
 * its present implementation does not support clipping or
 * double buffering. Incorporating these techniques would
 * improve the performance of instances of this class
 * substantially.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

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

```
public class DiagramOfSequence extends Canvas
{

```

```
DoSListener listener;
```

```
FTEd_ResourceRegistry rr;
CharacterRect[] rects;
char[] seq;
char widthDummy = 'A';

```

```
Font f;
FontMetrics fm;
int fontsize;

```

```
int xborder = 20, yborder = 40;
boolean autoSelect = true;

```

```
int ox, oy, w, h, ascent;
int lasti = -1;

```

```
Point selectedRegion, oldRegion;
```

```
Color bg = Color.black;
Color RED = Color.red.darker();

```

```
static final int PROTEIN = 0;
static final int DNA = 1;
int seqmode = PROTEIN;

```

```
public DiagramOfSequence(String sequence, DoSListener l)
{

```

```
    listener = l;
```

```
    rr = FTED_ResourceRegistry.instance();
```

```
    initialize(sequence);
```

```
    fontsize = 16;
    refontify();
    this.setBackground(bg);

```

```
}
```

```
private void initialize(String sequence)
{

```

```

seq = new char[sequence.length()];
seq = sequence.toCharArray();

rects = new CharacterRect[seq.length];

for(int i=0; i<seq.length; i++)
{
    rects[i] = new CharacterRect(this, String.valueOf(seq[i]),
getColor(seq[i]),bg);
}

public void paint(Graphics g)
{
    ox = xborder;
    oy = yborder;

    int x = ox, y = oy;

    fm = g.getFontMetrics(f);
    ascent = fm.getAscent();

    w = fm.charWidth(widthDummy);
    h = ascent + fm.getDescent();

    CharacterRect trect;

    for(int i = 0; i<rects.length; i++)
    {
        trect = rects[i];
        trect.reshape(x,y,w,h);
        trect.setAscent(ascent);

        trect.paint(g);

        x += w;
    }

    if(selectedRegion != null)
    {
        underlineRegion(g);
    }
}

```

```

protected void underlineRegion()
{
    Graphics g = this.getGraphics();
    underlineRegion(g);
}

protected void underlineRegion(Graphics g)
{
    g.setColor(RED);
    g.fill3DRect((selectedRegion.x)*w + ox,
oy + h + h/2,
w*(selectedRegion.y - selectedRegion.x + 1),
h/4,
false
);
}

protected void removeRegionUnderline(Point r)
{
    Graphics g = this.getGraphics();
    g.fillRect((selectedRegion.x)*w + ox,
oy + h + h/2,
w*(selectedRegion.y - selectedRegion.x + 1),
h/4
);
}

public void selectRegion(int first, int last)
{
    if(selectedRegion != null)
    {
        removeRegionUnderline(selectedRegion);
    }

    selectedRegion = new Point ( first, last );

    underlineRegion();

    if(autoSelect == true){
    try{

```

```

        for(int i = 0; i<first; i++)
        {
            rects[i].deselect();
        }
        for(int i = first; i<=last; i++)
        {
            rects[i].select();
        }
        for(int i = last + 1; i<rects.length; i++)
        {
            rects[i].deselect();
        }
    } catch (ArrayIndexOutOfBoundsException e) {e.printStackTrace();}

    repaint();
}

}

public void selectRegions(Point[] points)
{
    for(int i=0; i<rects.length; i++)
    {
        if(rects[i].isSelected())
        {
            rects[i].deselect();
        }
    }

    for(int j = 0; j<points.length; j++)
    {
        for(int i = points[j].x ; i< points[j].y; i++)
        {
            rects[i].select();
        }
    }
}

public int getXPositionOf(int index)
{
    return xborder + w*index;
}

```

```

public void clearAll()
{
    for(int i=0; i<rects.length; i++)
    {
        if(rects[i].isSelected())
        {
            rects[i].deselect();
        }
    }
    selectedRegion = null;
    repaint();
}

public boolean getAutoSelect()
{
    return autoSelect;
}

public void setAutoSelect(boolean t)
{
    autoSelect = t;
}

public Dimension preferredSize()
{
    Graphics g = this.getGraphics();
    fm = g.getFontMetrics(f);
    return new Dimension ( 2*xborder + seq.length*fm.charWidth(widthDummy),
                           2*yborder + fm.getAscent() +
                           fm.getDescent());
}

public Dimension minimumSize()
{
    return preferredSize();
}

public void setFSize(int value)
{
    fontsize = value;
    refontify();
}

```

```

        repaint();
    }

    public int getFSize()
    {
        return fontsize;
    }

    public void reset(String sequence)
    {
        initialize(sequence);
        repaint();
    }

    public void setSeqMode(int mode)
    {
        switch(mode)
        {
            case DNA:
                seqmode = DNA;
                break;
            case PROTEIN:
            default:
                seqmode = PROTEIN;
                break;
        }
    }

    protected void refontify()
    {
        f = new Font ("Courier", Font.PLAIN, fontsize);
        this.setFont(f);
    }

    //Event handling

    public boolean mouseDown(Event e, int x, int y)
    {
        CharacterRect cr = getCharacterRectAt(x,y);
        if(cr != null)
        {
            toggleSelect( cr );

```

```

            listener.registerSelect(lasti);
            return true;
        }
        return false;
    }

    protected void toggleSelect(CharacterRect cr)
    {
        if(cr.isSelected())
        {
            cr.deselect();
        }
        else
        {
            cr.select();
        }
        cr.paint();
    }

    protected CharacterRect getCharacterRectAt(int x, int y)
    {
        if(yWithinSequence(y))
        {
            lasti = (int)((double)(x - xborder)/(double)w );
            return rects[lasti];
        }
        return null;
    }

    protected boolean yWithinSequence(int y)
    {
        return ( (y > yborder) && (y < yborder + h) );
    }

    protected Color getColor(char c)
    {
        switch(seqmode)
        {
            case PROTEIN:
                return getResidueColor(c);
            case DNA:
                return getBaseColor(c);
        }
    }

```

```

        return null;
    }

    protected Color getResidueColor(char c)
    {
        return rr.getSequenceColors().getResidueColor(c);
    }

    protected Color getBaseColor(char c)
    {
        return rr.getSequenceColors().getBaseColor(c);
    }

    public int getLastIndex()
    {
        return lasti;
    }
}

```

```

/*
 * CLASS: DiagramOfSequenceGUI
 *
 * This class encapsulates a GUI macrocomponent that contains
 * buttons, text windows and a slider, and provides a scrollable
 * frame for the DiagramOfSequence. The buttons provide an
 * interface to some of the editing and viewing functions, the
 * text window dynamically displays information about the last
 * residue/base and the last region selected, while the slider
 * adjusts the font size of characters in the DiagramOfSequence.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;
import gjt.*;
import symantec.itools.awt.*;

```

```

public class DiagramOfSequenceGUI extends Frame
{

```

```

DoSGUIListener listener;
DiagramOfSequence dos;

```

```

ComponentScroller scroller = null;
Button    bu_setStart = null;
Button    bu_setEnd = null;
Button    bu_selectReg = null;
Button    bu_makeFeat = null;
Button    bu_clear = null;
Button    bu_customize = null;
Checkbox  cb_highlight = null;
TextField ta_res = null;
TextField ta_region = null;
Slider    sb_sizer = null;

```

```

Scrollbar hbarRef = null;

```

```

GridBagLayout gridbag = null;
gblHelper gbl = null;

```

```

public DiagramOfSequenceGUI(DiagramOfSequence dos, DoSGUIListener l)
{
    super("Sequence Diagram");
    this.dos = dos;
    listener = l;

    instantiateComponents();
    layoutComponents();
    setUpComponents();
    this.pack();
    this.show();
}

```

```

private void instantiateComponents()
{

```

```

    scroller = new ComponentScroller(dos);
    scroller.getViewport().setBackground(Color.black);
    hbarRef = scroller.getHorizontalScrollbar();

```

```

    bu_setStart = new Button("Set as Start");
    bu_setEnd = new Button("Set as End");
    bu_selectReg = new Button("Select Region");
    bu_makeFeat = new Button("Make Feature");
}

```

```

    bu_clear = new Button("Clear");
    bu_customize = new Button("Customize");

    cb_highlight = new Checkbox("Highlight Selections");

    int textfield_cols = 20;
    ta_res = new TextField(textfield_cols);
    ta_region = new TextField(textfield_cols);

    sb_sizer = new VerticalSlider();
}

private void setUpComponents()
{
    ta_res.setEditable(false);
    ta_region.setEditable(false);
    cb_highlight.setState(listener.getHighlightPreference());

    sb_sizer.setTickFreq(4);
    sb_sizer.setMinValue(-48);
    sb_sizer.setMaxValue(-8);
    sb_sizer.setValue(- listener.getFSize() );
    sb_sizer.resize(50,80);
    sb_sizer.setShowBorder(false);
}

private void layoutComponents()
{
    gridbag = new GridBagLayout();
    gbl = new gblHelper();

    Panel control_panel = layoutControlPanel();
    Panel scroller_panel = layoutScrollerPanel();

    Panel contain_all = new Panel();
    contain_all.setLayout(new BorderLayout());

    contain_all.add("Center", scroller_panel);
    contain_all.add("South", control_panel);

    Label title_label = new Label("Sequence Diagram", Label.CENTER);

    Box title_box = new Box(contain_all, title_label);

```

```

        this.add("Center", title_box);
    }

    private Panel layoutScrollerPanel()
    {
        Panel p1 = new Panel();
        p1.setLayout(gridbag);

        Insets myinsets = new Insets(0,0,0,0);

        gbl.constrain(p1, scroller, 0,0, 1,1, GridBagConstraints.BOTH,
GridBagConstraints.NORTHWEST,
        1.1, 1.1, myinsets);
        gbl.constrain(p1, new TestCanvas(0,120), 1,0, 1,1, GridBagConstraints.NONE,
GridBagConstraints.CENTER,
        0.0, 0.0, myinsets);

        return new EtchedBorder(p1);
    }

    private Panel layoutControlPanel()
    {
        Panel sizer_panel = layoutSizerPanel();
        Panel curr_panel = layoutCurrPanel();
        Panel view_panel = layoutViewPanel();
        Panel selection_panel = layoutSelectionPanel();

        Panel control_panel = new Panel();
        control_panel.setLayout(gridbag);

        Insets insets = new Insets(2,5,2,5);

        gbl.constrain(control_panel, sizer_panel, 0,0, 1,1,
GridBagConstraints.VERTICAL, GridBagConstraints.CENTER,
        0.0, 0.0, insets);
        gbl.constrain(control_panel, curr_panel, 1,0, 1,1,
GridBagConstraints.VERTICAL, GridBagConstraints.CENTER,
        0.0, 0.0, insets);
        gbl.constrain(control_panel, view_panel, 2,0, 1,1,
GridBagConstraints.VERTICAL, GridBagConstraints.CENTER,
        0.0, 0.0, insets);
    }

```

```

        gbl.constrain(control_panel, selection_panel, 3,0, 1,1,
GridBagConstraints.VERTICAL, GridBagConstraints.CENTER,
        0.0, 0.0, insets);

        return control_panel;
    }

    private Box layoutSizerPanel()
    {
        Insets insets = new Insets(0,0,0,0);

        Label la_bigger = new Label("bigger", Label.LEFT);
        Label la_smaller = new Label("smaller", Label.LEFT);

        Panel sizer_panel = new Panel();
        sizer_panel.setLayout(gridbag);

        gbl.constrain(sizer_panel, sb_sizer, 0,0, 1,2, GridBagConstraints.VERTICAL,
GridBagConstraints.WEST,
        0.0, 1.0, insets);
        gbl.constrain(sizer_panel, la_bigger, 1,0, 1,1, GridBagConstraints.NONE,
GridBagConstraints.NORTHWEST,
        0.0, 0.0, insets);
        gbl.constrain(sizer_panel, la_smaller, 1,1, 1,1, GridBagConstraints.NONE,
GridBagConstraints.SOUTHWEST,
        0.0, 0.0, insets);

        return new Box(sizer_panel, "Size");;
    }

    private Panel layoutCurrPanel()
    {
        Panel p1 = new Panel();
        p1.setLayout(new GridLayout(2,1));

        Box b1 = new Box(ta_res, "Current Res/Base");
        Box b2 = new Box(ta_region, "Current Region");

        p1.add(b1);
        p1.add(b2);

        return p1;
    }

```

```

private Panel layoutViewPanel()
{
    Panel p1 = new Panel();
    p1.setLayout(new GridLayout(3,1));

    p1.add(cb_highlight);
    p1.add(bu_customize);
    p1.add(bu_clear);

    return new Box(p1, "View");
}

private Box layoutSelectionPanel()
{
    Panel p1 = new Panel();
    p1.setLayout(new RowLayout());
    p1.add(bu_setStart);
    p1.add(bu_setEnd);

    Panel p2 = new Panel();
    p2.setLayout(new GridLayout(3,1));
    p2.add(p1);
    p2.add(bu_selectReg);
    p2.add(bu_makeFeat);

    return new Box(p2, "SelectRegion");
}

public void setCurrentRes(String resinfo)
{
    ta_res.setText(resinfo);
}

public void setCurrentRegion(String reginfo)
{
    ta_region.setText(reginfo);
}

public void clearCurrentRegion()
{
    ta_region.setText("");
}

```

```

public void scrollTo(int value)
{
    hbarRef.setValue(value);
    scroller.scrollTo(value, scroller.getVerticalScrollbar().getValue());
}

public boolean action(Event e, Object arg)
{
    if(e.target == bu_setStart)
    {
        listener.setStart();
        return true;
    }
    if(e.target == bu_setEnd)
    {
        listener.setEnd();
        return true;
    }
    if(e.target == bu_selectReg)
    {
        listener.selectRegion();
        return true;
    }
    if(e.target == bu_makeFeat)
    {
        listener.makeFeature();
        return true;
    }
    if(e.target == bu_clear)
    {
        listener.clear();
        return true;
    }
    if(e.target == bu_customize)
    {
        new CustomizeDoS(listener);
        return true;
    }
    if(e.target == cb_highlight)
    {
        listener.highlightSelections(cb_highlight.getState());
        return true;
    }
}

}
if(e.target == sb_sizer)
{
    listener.setFSize(- sb_sizer.getValue());
    scroller.invalidate();
    scroller.validate();
    scroller.repaint();
    return true;
}
return false;
}

public void die()
{
    this.hide();
    this.dispose();
}
}

}

/*
 * INTERFACE: DoFListener
 *
 * This interface defines services requested by the
 * DiagramOfFeatures and DiagramOfFeaturesGUI classes,
 * and is implemented by Hub in the present version of
 * J-Features..
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

public interface DoFListener extends FeatureStore, Locker
{
    public void edit(int index);
    public FeatureTableEntry delete(int index);
    public void showFeature(int index);
    public void quickShowFeature(int index);
    public void newFeature();
}

```

```

/*
 * INTERFACE: DoSGUIListener
 *
 * This interface defines the services requested by the
 * DiagramOfSequenceGUI class. In the present version of
 * J-Features, the DoSManager implements this interface
 * and provides these services.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

public interface DoSGUIListener
{
    public void setStart();
    public void setEnd();
    public void selectRegion();
    public void makeFeature();
    public void clear();
    public void highlightSelections(boolean h);
    public boolean getHighlightPreference();
    public void setFSize(int value);
    public int getFSize();
    public void applyColorChange();
}

```

```

/*
 * INTERFACE: DoSListener
 *
 * This interface defines the services requested by the
 * DiagramOfSequence class. In the present version of
 * J-Features, this interface is implemented by the
 * DoSManager.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

public interface DoSListener
{
    public void registerSelect(int index);
}

```

```

}

```

```

/*
 * CLASS: DoSManager
 *
 * This class encapsulates an object that coordinates the
 * DiagramOfSequence and DiagramOfSequenceGUI and provides
 * its parent with a single interface to the services of
 * both.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;

```

```

public final class DoSManager implements DoSGUIListener, DoSListener
{
    String sequence;
    int start;
    int end;
    DiagramOfSequence dos;
    DiagramOfSequenceGUI dosgui;
    DoSManagerListener listener;

    public DoSManager(DoSManagerListener l)
    {
        listener = l;
    }

    public void setSequence(String seq)
    {
        sequence = seq;
        if(dos != null)
        {
            dos.reset(sequence);
            //something
        }
        else
        {
            dos = new DiagramOfSequence(sequence, this);
        }
    }
}

```

```

        dosgui = new DiagramOfSequenceGUI( dos, this);
    }
    //dos = new DiagramOfSequence(sequence, this);
    //dosgui = new DiagramOfSequenceGUI( dos, this);
}

public void selectRegion(int numstart, int numend)
{
    this.start = numstart-1;
    this.end = numend -1;

    this.selectRegion();
}

//implement DoSGUIListener interface

public void setStart()
{
    start = dos.getLastIndex();
    dosgui.clearCurrentRegion();
}

public void setEnd()
{
    end = dos.getLastIndex();
    dosgui.clearCurrentRegion();
}

public void selectRegion()
{
    if(regionIsValid())
    {
        int offset = 2;
        dosgui.scrollTo(dos.getXPositionOf(start - 2));
        dos.selectRegion(start, end);
        dos.repaint();
        dosgui.setCurrentRegion( (start + 1) + " to " + (end + 1) );
    }
}

public void makeFeature()
{
    if(regionIsValid())

```

```

        {
            listener.newFeature( new Point( (start + 1), (end + 1) ) );
        }
    }

public void clear()
{
    dos.clearAll();
}

public void highlightSelections(boolean h)
{
    dos.setAutoSelect(h);
}

public boolean getHighlightPreference()
{
    return dos.getAutoSelect();
}

public void setFSize(int value)
{
    System.out.println("Value : " + value);
    dos.setFSize(value);
}

public int getFSize()
{
    return dos.getFSize();
}

public void applyColorChange()
{
    dos.reset(sequence);
}

//implement DoSListener interface

public void registerSelect(int index)
{
    dosgui.setCurrentRes( (index + 1) + " (" + sequence.substring(index, index + 1)
+ " )" );
}

```

```

private boolean regionsValid()
{
    return ((start <= end) && (start > -1));
}

public void die()
{
    dosgui.die();
}
}

```

```

/*
 * INTERFACE: DoSManagerListener
 *
 * This interface defines the services requested by the
 * DoSManager. In the present version of J-Features, it
 * is implemented by Hub.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;

public interface DoSManagerListener
{
    public void newFeature(Point location);
}

```

```

/*
 * CLASS: DummyCanvas
 *
 * This class defines an object that acts as a 'dummy' or
 * 'blank' component. The DummyCanvas is a simple object
 * that takes up the required amount of space in a
 * container's layout. DummyCanvas, and the identical
 * TestCanvas, are used as spacers in grid and gridbag
 * layouts.

```

```

 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;

public class DummyCanvas extends Canvas
{
    int w,h;

    public DummyCanvas(int w, int h)
    {
        this.w = w;
        this.h = h;
    }

    public DummyCanvas()
    {
        this(200,100);
    }

    public Dimension preferredSize()
    {
        return new Dimension(w,h);
    }

    public Dimension minimumSize()
    {
        return preferredSize();
    }
}

```

```

/*
 * CLASS: FTED_ResourceRegistry
 *
 * This class defines an object that acts as a global information
 * store for J-Features. Some of the information held in this
 * class might be more appropriately held outside the global space.
 * Similarly, there are some data areas in other classes that would
 * be more appropriately placed within this class.

```

```

*
* FTed_ResourceRegistry has a protected constructor invoked by a
* static method that ensures no more than one instance of this
* class can ever exist. Many objects can obtain a reference to
* this class independently. Because of this, the resource registry
* can only be destroyed explicitly through the die() method. Note
* that the die() method is not failsafe and may not even be
* particularly effective.
* @author William Valdar
* @version 1.0 (September 1997)
*/

import java.util.*;

public class FTed_ResourceRegistry
{
    static DBSeqInfo currentSequence;

    static Vector standardTypes;

    static SequenceColors sequenceColors;

    static Hashtable typesGProps;

    static Hashtable hashtable;

    static FeatureColorsROR featureColorsROR;

    protected static FTed_ResourceRegistry registry;

    protected FTed_ResourceRegistry()
    {
        hashtable = new Hashtable();
    }

    public synchronized static FTed_ResourceRegistry instance()
    {
        if(registry == null)
        {
            registry = new FTed_ResourceRegistry();
        }
        return registry;
    }
}

```

```

}

public synchronized static void die()
{
    if(registry != null)
    {
        registry = null;
    }
}

public synchronized void addProperty(String name, Object value)
{
    hashtable.put(name, value);
}

public synchronized Object getProperty(String name)
{
    return hashtable.get(name);
}

public synchronized boolean hasPropertyName(String name)
{
    return hashtable.containsKey(name);
}

public synchronized void addTypeGProperties (String name, TypeGProperties value)
{
    typesGProps.put(name, value);
}

public synchronized TypeGProperties getTypeGProperties(String name)
{
    return (TypeGProperties)typesGProps.get(name);
}

public synchronized boolean hasTypeGPropertiesFor(String name)
{
    return typesGProps.containsKey(name);
}

public Enumeration getAllTypes()
{
    return typesGProps.keys();
}

```

```

    }

    public synchronized void replaceTypeGProperties(String name, TypeGProperties tgp)
    {
        typesGProps.remove(name);
        typesGProps.put(name, tgp);
    }

    public synchronized void setCurrentSequence(DBSeqInfo dbsi)
    {
        currentSequence = dbsi;
    }

    public DBSeqInfo getCurrentSequence()
    {
        return currentSequence;
    }

    public synchronized void setStandardTypes(Vector st)
    {
        standardTypes = st;

        initializeTypesGProps();
    }

    public synchronized Vector getStandardTypes()
    {
        return standardTypes;
    }

    private void initializeTypesGProps()
    {
        int vsize = standardTypes.size();

        typesGProps = new Hashtable(vsize + 10);

        for(int i = 0; i<vsize; i++)
        {
            typesGProps.put( (String) standardTypes.elementAt(i), new
TypeGProperties() );
        }
    }

```

```

    public synchronized void setSequenceColors(SequenceColors sc)
    {
        sequenceColors = sc;
    }

    public synchronized SequenceColors getSequenceColors()
    {
        return sequenceColors;
    }

    public synchronized void setFeatureColorsROR(FeatureColorsROR fcror)
    {
        featureColorsROR = fcror;
    }

    public synchronized FeatureColorsROR getFeatureColorsROR()
    {
        return featureColorsROR;
    }
}

/*
 * CLASS: FeatureColorsROR
 *
 * This simple class constructs a TwoWayHashTable containing
 * information about allowed colours for the features in the
 * DiagramOfFeatures, and the names of those colours. It would
 * be more appropriate for the services provided by this class
 * to be embodied in initiation procedures carried out within
 * either the DiagramOfFeatures class or the Hub.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.awt.*;
import java.util.*;

public class FeatureColorsROR
{
    TwoWayHashtable availableColorHash;

```

```

Vector colorNames;

public FeatureColorsROR()
{
    TwoWayHashtable h = new TwoWayHashtable();

    h.put("black",      Color.black);
    h.put("blue",       Color.blue);
    h.put("cyan",       Color.cyan);
    h.put("dark gray",  Color.darkGray);
    h.put("gray",       Color.gray);
    h.put("green",      Color.green);
    h.put("light gray", Color.lightGray);
    h.put("magenta",    Color.magenta);
    h.put("orange",     Color.orange);
    h.put("pink",       Color.pink);
    h.put("red",        Color.red);
    h.put("white",      Color.white);
    h.put("yellow",     Color.yellow);

    Vector v = MyUtils.getVectorOfEnumeration( h.keys() );
    MyUtils.sortVectorOfStringsFwd( v );

    availableColorHash = h;
    colorNames = v;
}

public TwoWayHashtable getAvailableColorHash()
{
    return availableColorHash;
}

public Vector getColorNames()
{
    return colorNames;
}
}

```

```

/*
 * CLASS: FeatureInputDialog
 *
 * This class encapsulates a GUI dialog box that allows the

```

```

 * user to enter information about a feature. Inconsistencies
 * in the users input are handled by this class, rather than
 * the parent, and result in appropriate dialog boxes being
 * spawned.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;
import javax.swing.*;
import java.util.*;

```

```

public class FeatureInputDialog extends JFrame
{

```

```

    FeatureTableEntry fte;
    FeatureTableEntry original_fte;
    FeatureInputDialogListener listener;

```

```

    TextField tf_start;
    TextField tf_end;
    TextField tf_type;
    Choice ch_type;
    TextArea ta_q;
    TextArea ta_c;

```

```

    Button bu_accept;
    Button bu_cancel;

```

```

    GridBagLayout gridbag;
    GridBagConstraints gbl;

```

```

    boolean existingFeature;

```

```

    boolean debug = true;
    String CUSTOM_MARK = "[CUSTOM:]";
    String UNDEFINED_MARK;

```

```

    public FeatureInputDialog(FeatureTableEntry fte, FeatureInputDialogListener l, boolean
existingFeature)
    {
        super("Feature Input");

```

```

    this.existingFeature = existingFeature;
    this.fte = fte;
    original_fte = (FeatureTableEntry) fte.clone();
    listener = l;
    UNDEFINED_MARK = fte.getDEFAULT_TYPE();

    instantiateComponents();
    initializeComponents();
    layoutComponents();
    this.pack();
    this.setResizable(false);
    this.show();
}

private void instantiateComponents()
{
    int locsize = 5;
    tf_start = new TextField(locsize);
    tf_end = new TextField(locsize);

    int typesize = 15;
    tf_type = new TextField(typesize);

    ch_type = new Choice();

    int rows = 6;
    int columns = 40;
    ta_q = new TextArea(rows, columns);
    ta_c = new TextArea(rows, columns);

    bu_accept = new Button("Accept");
    bu_cancel = new Button("Cancel");
}

private void layoutComponents()
{
    gridbag = new GridBagLayout();
    gbl = new gblHelper();

    Panel locp = layoutLocationPanel();
    Panel tp = layoutTypePanel();
    Panel qcp = layoutQCPanel();
    Panel bp = layoutButtonPanel();

    Panel contain_all = new Panel();
    contain_all.setLayout(gridbag);

    Insets myinsets = new Insets(0,0,0,0);

    gbl.constrain(contain_all, locp, 0,0, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);
    gbl.constrain(contain_all, tp, 0,1, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);
    gbl.constrain(contain_all, qcp, 0,2, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);
    gbl.constrain(contain_all, bp, 0,3, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);

    Box titlebox = new Box(contain_all, "Enter Feature");

    this.add("Center", titlebox);
}

private Panel layoutLocationPanel()
{
    Panel p1 = new Panel();
    p1.setLayout(new GridLayout(1,4, 5,0));

    p1.add(new Label("Start:", Label.LEFT));
    p1.add(tf_start);
    p1.add(new Label("End:", Label.LEFT));
    p1.add(tf_end);

    Box b1 = new Box(p1, "Location");

    return b1;
}

private Panel layoutTypePanel()
{
    Panel p1 = new Panel();
    p1.setLayout(new GridLayout(1,2,5,0));

```

```

        p1.add(ch_type);
        p1.add(tf_type);

        Box b1 = new Box(p1, "Type / Key");

        return b1;
    }

    private Panel layoutQCPanel()
    {
        Panel p1 = new Panel();
        p1.setLayout(gridbag);

        Label la_q = new Label("Qualifiers :", Label.LEFT);
        Label la_c = new Label("Comments :", Label.LEFT);

        Insets myinsets = new Insets(0,0,0,0);

        gbl.constrain(p1, la_q, 0,0, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.WEST,
                    0.0, 0.0, myinsets);
        gbl.constrain(p1, ta_q, 0,1, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);
        gbl.constrain(p1, la_c, 0,2, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.WEST,
                    0.0, 0.0, myinsets);
        gbl.constrain(p1, ta_c, 0,3, 1,1, GridBagConstraints.HORIZONTAL,
GridBagConstraints.CENTER,
                    0.0, 0.0, myinsets);

        return p1;
    }

    private Panel layoutButtonPanel()
    {
        Panel p1 = new Panel();
        p1.setLayout(new GridLayout(1,3));

        p1.add(bu_accept);
        p1.add(new DummyCanvas(0,30)); //lightweight dummy
        p1.add(bu_cancel);

        Panel p2 = new Panel();
        p2.setLayout(new GridLayout(2,1));

        Separator sep = new Separator();

        p2.add(sep);
        p2.add(p1);

        return p2;
    }

    private void initializeComponents()
    {
        Font f = null;

        try{
            Font y = this.getParent().getFont();
            f = new Font("Courier", Font.PLAIN, y.getSize() + 4);
        }
        catch(Exception e)
        {
            if(debug)
            {
                System.out.println("***Null parent font***");
                e.printStackTrace();
            }
            f = new Font("Courier", Font.PLAIN, 16);
        }
        tf_start.setFont(f);
        tf_end.setFont(f);
        tf_type.setFont(f);
        ta_q.setFont(f);
        ta_c.setFont(f);

        setUpChoices(); //Do not swap without changing
        setUpText(); //other code
    }

    private void setUpChoices()
    {
        Vector stv = FTED_ResourceRegistry.instance().getStandardTypes();
    }

```

```

MyUtils.choiceItemsFromStringVector(ch_type, stv);
ch_type.addItem(CUSTOM_MARK);
String mytype = fte.getType();
if( stv.contains( mytype ) )
{
    ch_type.select( mytype );
}
else
{
    ch_type.select(CUSTOM_MARK);
}
}

private void setUpText()
{
    tf_start.setEditable(true);
    tf_start.setText( Integer.toString(fte.getLocation().x) );

    tf_end.setEditable(true);
    tf_end.setText( Integer.toString(fte.getLocation().y) );

    if(ch_type.getSelectedItem().equals(CUSTOM_MARK)) //one reason
why setUpChoices must go first
    {
        tf_type.setEditable(true);
    }
    else
    {
        tf_type.setEditable(false);
    }
    tf_type.setText( fte.getType() );

    ta_q.setEditable(true);
    if(fte.qualifiersExist())
    {
        ta_q.setText( MyUtils.VectorOfStringsToString( fte.getQualifiers() )
);
    }

    ta_c.setEditable(true);

    if(fte.commentsExist())
    {
        ta_c.setText( MyUtils.VectorOfStringsToString( fte.getQualifiers() )
);
    }
}

public boolean accept()
{
    try{
        int start = Integer.parseInt(tf_start.getText());
        int end = Integer.parseInt(tf_end.getText());
        if(start > end) throw new Exception("start before end");
        else fte.setLocation(new Point(start, end));

        String string = null; Vector stringvector;

        string = tf_type.getText().trim();
        if(string.equals(UNDEFINED_MARK) || string.equals("")) throw
new Exception("no type");

        fte.setType(string);

        string = ta_q.getText();
        stringvector = MyUtils.StringToVector(string);
        if( MyUtils.ridVectorOfEmptyStrings( stringvector ).isEmpty() ==
false )
        {
            fte.setQualifiers( stringvector );
        }

        string = ta_c.getText();
        stringvector = MyUtils.StringToVector(string);
        if( MyUtils.ridVectorOfEmptyStrings( stringvector ).isEmpty() ==
false )
        {
            fte.setComments( stringvector );
        }
    }
    catch(NumberFormatException e)
    {
        if(debug) {e.printStackTrace();}
        new InfoDialog(this, "Attention",

```

```

        "The start and end boxes are unfilled"
        + "\nor contain non-integer characters."
        + "\nPlease re-enter the positions using whole
numbers only.",
        false);
    return false;
}
catch(Exception e)
{
    if(debug) {e.printStackTrace();}
    if(e.getMessage().equals("start before end"))
    {
        new InfoDialog(this, "Attention",
            "The start position is after the end position."
            + "\nPlease re-enter the positions so that the
start"
            + "\nis before the end.",
            false);
    }
    if(e.getMessage().equals("no type"))
    {
        new InfoDialog(this, "Attention",
            "A key description has not been supplied."
            + "\nPlease enter either a key from the"
            + "\nlist or a custom key.",
            false);
    }
    return false;
}

listener.add(fte);
listener.makeInputFormNull();
return true;
}

public void cancel()
{
    if(existingFeature){listener.add(original_fte);}
    listener.makeInputFormNull();
    die();
}

public void die()

```

```

    {
        this.hide();
        this.dispose();
    }

    public boolean action(Event e, Object arg)
    {
        if(e.target == ch_type)
        {
            if(arg.equals(CUSTOM_MARK))
            {
                tf_type.setText("");
                tf_type.setEditable(true);
                return true;
            }
            else
            {
                tf_type.setText(arg.toString());
                tf_type.setEditable(false);
                return true;
            }
        }
        if(e.target == bu_accept)
        {
            if(accept() == true)
            {
                die();
            }
            return true;
        }
        if(e.target == bu_cancel)
        {
            cancel();
            return true;
        }
        return false;
    }
}
/*
 * INTERFACE: FeatureInputFormListener
 */

```

```

* This interface defines the services requested by the
* FeatureInputForm class. In the current version of
* J-Features, this interface is implemented by Hub only.
* @author William Valdar
* @version 1.0 (September 1997)
*/

```

```

public interface FeatureInputFormListener
{
    public void add(FeatureTableEntry fte);
    public void makeInputFormNull();
}

```

```

/*
* INTERFACE: FeatureStore
*
* This interface defines methods provided by a class that
* can access the main store of data about the feature
* table.
* @author William Valdar
* @version 1.0 (September 1997)
*/

```

```

public interface FeatureStore
{
    public FeatureTableEntry getFeatureAt(int index);
    public boolean hasFeatureAt(int index);
}

```

```

/*
* CLASS: FeatureTableEntry
*
* This abstract class is the subclass of SwissFTEntry and defines
* an object that hold information about a feature in a feature
* table. It was originally intended that subclasses should implement
* these methods in a database-specific manner. However, these are
* largely 'getter' and 'setter' methods which are likely to have

```

```

* the same simple implementation whatever the database context. If
* J-Features is revised, this class should either be made
* non-abstract and the only class used to store feature data, or
* it should remain abstract but implement some or all of its
* methods.
* @author William Valdar
* @version 1.0 (September 1997)
*/

```

```

import java.awt.*;
import java.util.*;

```

```

public abstract class FeatureTableEntry implements Cloneable
{

```

```

    protected Point location;
    protected String type;

```

```

    public abstract void setType(String type);
    public abstract String getType();
    public abstract String getDEFAULT_TYPE();

```

```

    public abstract void setLocation(Point location);
    public abstract Point getLocation();

```

```

    public abstract boolean qualifiersExist();
    public abstract boolean commentsExist();

```

```

    public abstract void setQualifiers(Vector qualifiers);
    public abstract Vector getQualifiers();
    public abstract void setComments(Vector comments);
    public abstract Vector getComments();

```

```

    public abstract String getFormat();
    public abstract String toString();
    public abstract boolean equals(Object obj);
    public abstract boolean isUpstreamOf(FeatureTableEntry fte);
    public abstract Object clone();

```

```

}

```

```

/*
 * CLASS: FeatureTableEntryPredicate
 *
 * This class implements the jgl.BinaryPredicate interface and
 * provides a predicate that allows FeatureTableEntry objects
 * to be sorted using the jgl implementation of the quicksort
 * algorithm. FeatureTableEntry objects are evaluated according
 * to position, those farthest upstream being given priority.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import COM.objectspace.jgl.*;

public final class FeatureTableEntryPredicate implements BinaryPredicate
{
    public boolean execute(Object first, Object second)
    {
        return ((FeatureTableEntry) first).isUpstreamOf( (FeatureTableEntry) second);
    }
}

/*
 * CLASS: FrameOkayCancelDialog
 *
 * This class encapsulates a GUI dialog box with 'okay' and
 * 'cancel' buttons. Subclasses override the empty okay()
 * and cancel() methods so that they do something useful.
 * This class extends Frame, rather than the more obvious
 * Dialog, because, at the time of writing, Dialog's modal
 * function caused display problems on some platforms. As an
 * alternative, FrameOkayCancelDialog supports a pseudo-modal
 * mechanism through its parent's Locker interface.
 * Unfortunately, Hub's implementation of this interface, and
 * therefore the modal function supported by this class, is
 * not perfect. This class supports parents implementing the
 * Locker interface.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;
import gjt.*;

public class FrameOkayCancelDialog extends Frame
{
    private Button bu_okay;
    private Button bu_cancel;
    private MultiLineLabel message;
    private String subtitle;
    private Locker locker;
    private boolean modal;
    public int buttonwidth = 0;
    public int buttonheight = 30;

    public FrameOkayCancelDialog(Locker l, boolean modal, String subtitle, String mess)
    {
        super("Question");
        locker = l;
        this.modal = modal;
        this.subtitle = subtitle;
        this.message = new MultiLineLabel(mess);

        bu_okay = new Button("Okay");
        bu_cancel = new Button("Cancel");

        layoutComponents();

        if(modal){locker.lock();}

        this.show();
    }

    private void layoutComponents()
    {
        Panel p1 = new Panel();
        p1.setLayout(new GridLayout(1,3));

        p1.add(bu_okay);
        p1.add(new TestCanvas(buttonwidth, buttonheight));
        p1.add(bu_cancel);
    }
}

```

```

Panel bp = new Panel();
bp.setLayout(new GridLayout(2,1));

bp.add(new Separator());
bp.add(p1);

Panel p2 = new Panel();
p2.setLayout(new BorderLayout());

p2.add("Center", message);
p2.add("South", bp);

Box b1 = new Box(p2, subtitle);
this.add("Center", b1);
this.pack();
this.setResizable(false);
}

public boolean action(Event e, Object arg)
{
    if(e.target == bu_okay)
    {
        if(modal){locker.unlock();}
        okay();
        die();
        return true;
    }
    if(e.target == bu_cancel)
    {
        if(modal){locker.unlock();}
        cancel();
        die();
        return true;
    }
    return false;
}

//Overridden methods

public void okay()
{}

public void cancel()

```

```

}

protected void die()
{
    this.hide();
    this.dispose();
}

}

/*
 * CLASS: GraphicFeatureTestCanvas
 *
 * This class encapsulates an interactive graphic that
 * represents a sample feature in the DiagramOfFeatures.
 * The graphic allows the user to preview the appearance
 * of a feature as they use CustomizeDoF to customize the
 * shape and colour scheme in the DiagramOfFeatures.
 * GraphicFeatureTestCanvas is a subclass of canvas that
 * paints a MyDrawnShape, which it highlights on mouse
 * down events.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.awt.*;

public class GraphicFeatureTestCanvas extends Canvas
{
    MyDrawnShape mdshape;
    TypeGProperties tgp;
    boolean clickedflag;
    static final int NOEGGS = 0;
    static final int EGGSONLY = 1;
    static final int EGG = 2;
    int preference = EGG;

    Color background_color = Color.white;
    Color sequence_color = Color.black;
    Rectangle mybounds;
    int ox, oy, maxw, maxh;
    int borderwidth = 5;

```

```

int featurelength = 30;

public GraphicFeatureTestCanvas(TypeGProperties tgp, int preference)
{
    this.setBackground(background_color);

    this.preference = preference;

    mdshape = new MyDrawnShape(this);

    this.tgp = tgp;
}

public void paint(Graphics g)
{
    refreshScalers();

    g.setColor(sequence_color);
    g.fillRect(ox,oy-3,maxw,6);

    if(tgp.visible)
    {
        try{
            mdshape.setFillColor(tgp.fill_color.darker());
            mdshape.setLineColor(tgp.line_color);

            switch(preferance)
            {
                case NOEGGS:
                    if(tgp.shape != MyDrawnShape.EGGTIMER)
                    {mdshape.setShape(tgp.shape);}
                    break;
                case EGGSONLY:
                    mdshape.setShape(MyDrawnShape.EGGTIMER);
                    break;
                case EGGS:
                default:
                    mdshape.setShape(tgp.shape);
                    break;
            }
        }
        if(mdshape.getShape() == MyDrawnShape.EGGTIMER)
        {

```

```

            mdshape.reshape(ox + (maxw-DiagramOfFeatures.eggtimerW)/2,
                oy - DiagramOfFeatures.eggtimerH,
                DiagramOfFeatures.eggtimerW,
                2*DiagramOfFeatures.eggtimerH);
        }
        else
        {
            mdshape.reshape(ox + (maxw-featurelength)/2, oy - tgp.height,
                featurelength, 2*tgp.height);
        }
        mdshape.paint();
    }
    catch(NullPointerException e){e.printStackTrace();}
}

public boolean mouseDown(Event e, int x, int y)
{
    if(mdshape.inside(x,y))
    {
        highlightFeature();
        clickedflag = true;
        return true;
    }
    return false;
}

public boolean mouseUp(Event e, int x, int y)
{
    if(clickedflag == true)
    {
        repaint();
        clickedflag = false;
        return true;
    }
    else return false;
}

public void setTypeGProperties(TypeGProperties tgp)
{
    this.tgp = tgp;
    repaint();
}

```

```

private void highlightFeature()
{
    Color fcolor = mdshape.getFillColor();
    Color lcolor = mdshape.getLineColor();
    mdshape.setFillColor(fcolor.brighter());
    mdshape.paint();
    mdshape.setFillColor(fcolor);
}

private void refreshScalers()
{
    mybounds = this.bounds();
    ox = mybounds.x + borderwidth;
    oy = mybounds.height/2;
    maxw = mybounds.width-(2*ox);
    maxh = mybounds.height/2-(mybounds.y+borderwidth);
}

public Dimension getPreferredSize()
{
    return new Dimension(100,100);
}

public Dimension preferredSize()
{
    return getPreferredSize();
}

public Dimension minimumSize()
{
    return getMinimumSize();
}

public Dimension getMinimumSize() //is this really necessary?
{
    return getPreferredSize();
}
}

```

```

/*
 * CLASS: Hub
 *
 * This class defines J-Features' central module. Hub coordinates
 * the activities of the other high level classes, e.g.,
 * DiagramOfFeatures, DoSManager, and contains the only direct
 * reference to the DBSeqInfo data store. Representing the highest
 * level of control, Hub is probably the most likely class to change
 * if J-Features is extended. Hub supports the Locker interface.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.awt.*;
import java.util.*;
import java.applet.Applet;

public class Hub extends Frame implements MyIOListener,
    DoFListener,
    FeatureInputFormListener,
    InputFormDialogListener,
    DoSManagerListener,
    FeatureStore,
    HubGUIListener
{
    Applet applet;
    SingleChildRestriction scr;
    HubGUI hubgui;
    MyIO myIO;
    DiagramOfFeatures dof;
    DiagramOfFeaturesGUI dofgui;
    DoSManager dosManager;

    FeatureInputForm featureInputForm = null;

    DBSeqInfo dbsi;
    FeatureTableEntry[] features = null;

    int seqlength = 500;
    boolean locked = false;
}

```

```
boolean debug = false;
Frame debugFrame = null;
```

```
Button bu_enter = null;
Button bu_translate = null;
Button bu_show = null;
Button bu_close = null;
```

```
TextArea ta = null;
```

```
public Hub (Applet applet, SingleChildRestriction scr)
{
```

```
    this.applet = applet;
    this.scr = scr;
```

```
    FTed_ResourceRegistry rr = FTed_ResourceRegistry.instance();
    SwissDetails sd = new SwissDetails();
    rr.setStandardTypes( sd.getStandardTypes());
    rr.setFeatureColorsROR (new FeatureColorsROR());
    rr.setSequenceColors(new SequenceColors());
```

```
    hubgui = new HubGUI(applet, this);
```

```
    myIO = new MyIO(this, new SwissProtTranslator());
```

```
    dof = new DiagramOfFeatures(this);
```

```
    dofgui = new DiagramOfFeaturesGUI(dof, this);
```

```
    if(debug)
    {
        instantiateComponents();
        layoutComponents();
        this.pack();
        this.show();
    }
}
```

```
private void instantiateComponents()
{
```

```
    bu_enter = new Button("Enter");
    bu_translate = new Button("Translate");
    bu_show = new Button("Show");
    bu_close = new Button("Close");
```

```
    ta = new TextArea(20,40);
    ta.setEditable(false);
```

```
}
```

```
private void layoutComponents()
```

```
{
```

```
    this.setLayout(new FlowLayout());
    this.add(bu_enter);
    this.add(bu_translate);
    this.add(bu_show);
    this.add(bu_close);
    this.add(ta);
```

```
}
```

```
//implement Locker interface
```

```
public void lock()
```

```
{
```

```
    this.locked = true;
```

```
}
```

```
public void unlock()
```

```
{
```

```
    this.locked = false;
```

```
}
```

```
//implements?
```

```
public void newFeature()
```

```
{
```

```
    if(this.locked) return;
    if(dbsi == null) return;
```

```
    clearLastForm();
```

```
    featureInputForm = new FeatureInputForm( newEntry(), this, false);
```

```
}
```

```

public void newFeature(Point location)
{
    if(this.locked) return;
    if(dbsi == null) return;

    clearLastForm();
    featureInputForm = new FeatureInputForm( newEntry(location), this, false);
}

private FeatureTableEntry newEntry()
{
    return new SwissFTEntry();
}

private FeatureTableEntry newEntry(Point loc)
{
    return new SwissFTEntry(loc);
}

//implement MyIOListener interface

public void setDBSeqInfo(DBSeqInfo dbsi)
{
    cleanse();

    this.dbsi = dbsi;
    seqlength = dbsi.getSequence().length();
    if(dbsi.hasFeatures())
    {
        resetFeatureArray();
        resetDiagramOfFeatures();
    }
    else
    {
        features = null;
    }
    resetDiagramOfSequence();
}

private void resetDiagramOfSequence()
{
    if(dosManager == null)
    {

```

```

        dosManager = new DoSManager(this);
    }
    dosManager.setSequence(dbsi.getSequence());
}

//implement FeatureStore interface

public FeatureTableEntry getFeatureAt(int index)
{
    if(dbsi.hasFeatures())
    {
        return (FeatureTableEntry) dbsi.getFeatures().elementAt(index);
    }
    return null;
}

public boolean hasFeatureAt(int index)
{
    if(features != null)
    {
        return features[index] != null;
    }
    return false;
}

//implement FeatureInputFormListener interface

public void add( FeatureTableEntry fte)
{
    if(this.locked) return;

    if(dbsi.hasFeatures())
    {
        Vector v = dbsi.getFeatures();
        v.addElement(fte);
    }
    else
    {
        Vector v = new Vector();
        v.addElement(fte);
        dbsi.setFeatures( v );
    }
    resetFeatureArray();
}

```

```

        resetDiagramOfFeatures();
    }

    public void makeInputFormNull()
    {
        featureInputForm = null;
    }

    //implement DoFListener interface

    public void edit(int index)
    {
        if(this.locked) return;

        if(features != null)
        {
            clearLastForm();
            featureInputForm = new FeatureInputForm( delete(index), this, true
);
        }
    }

    public FeatureTableEntry delete(int index)
    {
        if(this.locked) return null;

        FeatureTableEntry removed = null;
        if(dbsi != null)
        {
            if(dbsi.hasFeatures())
            {
                Vector v = dbsi.getFeatures();
                removed = (FeatureTableEntry) v.elementAt(index);
                v.removeElementAt(index);
                resetFeatureArray();
                resetDiagramOfFeatures();
            }
        }
        return removed;
    }

    private void clearLastForm()
    {

```

```

        if(featureInputForm != null)
        {
            new InputFormDialog(this, true, "Warning",
                "Changes to the feature currently\nbeing
edited will be lost!",
                this);
        }
    }

    //implement InputFormDialogListener interface

    public void cancelCurrentForm()
    {
        featureInputForm.cancel();
        featureInputForm = null;
    }

    private void cleanse()
    {
        if(featureInputForm != null)
        {
            featureInputForm.die();
            featureInputForm = null;
        }
    }

    private void resetFeatureArray()
    {
        features = null;
        Vector v = dbsi.getFeatures();
        features = new FeatureTableEntry[v.size()];
        v.copyInto(features);
    }

    private void resetDiagramOfFeatures()
    {
        dof.setInfo(features, seqlength);
    }

    public void showFeature(int index)
    {
        if(features != null)
        {

```

```

        FeatureTableEntry fte = features[index];
        new MyViewer(fte.toString(), "Show Feature",
                    "Feature", 20, 30);
        Point p = fte.getLocation();
        dosManager.selectRegion(p.x, p.y);
    }
}

public void quickShowFeature(int index)
{
    if(dofgui != null && features != null)
    {
        FeatureTableEntry fte = features[index];
        Point p = fte.getLocation();
        String s = new String(fte.getType());
        s += " (" + p.x + "-" + p.y + ")";
        dofgui.setCurrentFeatureBox(s);
    }
}

//implement HubGUIListener interface

public void open()
{
    myIO.receive();
}

public void close()
{
    hubgui.die();
    dofgui.die();
    if(dosManager != null) {dosManager.die();}
    FTED_ResourceRegistry.die();
    this.die();
}

public void translate()
{
    if(dbsi != null)
    {
        myIO.give((DBSeqInfo)dbsi.clone());
    }
}

```

```

public void die()
{
    scr.setChildToNull();
}

//Event handling

public boolean action(Event e, Object arg)
{
    if(debug){
        if(e.target == bu_enter)
        {
            open();
            return true;
        }
        if(e.target == bu_translate)
        {
            translate();
            return true;
        }
        if(e.target == bu_show)
        {
            if(dbsi != null)
            {
                ta.setText(dbsi.toString());
            }
            return true;
        }
        if(e.target == bu_close)
        {
            this.hide();
            this.dispose();
            return true;
        }
        return false;
    }
    else return false;
}

```

```

/*
 * CLASS: HubGUI
 *
 * This class encapsulates a GUI macrocomponent that is spawned
 * by Hub and provides an interface for simple input/ output
 * control.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.awt.*;
import java.applet.Applet;

public class HubGUI extends Frame
{
    protected MenuItem mi_open;
    protected MenuItem mi_save;
    protected MenuItem mi_close;
    protected MenuItem mi_about;

    protected Canvas canvas;
    private String filename = "J-Features.gif";
    protected Applet applet;
    protected HubGUIListener listener;

    public HubGUI(Applet applet, HubGUIListener l)
    {
        super("J - Features");
        this.applet = applet;
        listener = l;
        instantiateComponents();
        layoutComponents();
        this.show();
    }

    private void instantiateComponents()
    {
        canvas = new ImageFromFileCanvas(applet, filename, Color.white);

        mi_open = new MenuItem("Open");
        mi_save = new MenuItem("Save/Preview");

```

```

        mi_close = new MenuItem("Close");
        mi_about = new MenuItem("About");
    }

    private void layoutComponents()
    {
        Menu m_file = new Menu("File");
        m_file.add(mi_open);
        m_file.add(mi_save);
        m_file.addSeparator();
        m_file.add(mi_close);

        Menu m_help = new Menu("Help");
        m_help.add(mi_about);

        MenuBar menubar = new MenuBar();
        menubar.add(m_file);
        menubar.add(m_help);
        menubar.setHelpMenu(m_help);

        this.setMenuBar(menubar);
        this.add("Center", canvas);
        this.pack();
    }

    public boolean action(Event e, Object arg)
    {
        if(e.target == mi_open)
        {
            listener.open();
            return true;
        }
        if(e.target == mi_save)
        {
            listener.translate();
            return true;
        }
        if(e.target == mi_close)
        {
            listener.close();
            return true;
        }
        if(e.target == mi_about)

```

```

        {
            new InfoDialog(this, "About", "J - Features\nBy William S.J. Valdar
(Sept 1997)", false);
            return true;
        }
        return false;
    }

    public void die()
    {
        this.hide();
        this.dispose();
    }
}

```

```

/*
 * INTERFACE: HubGUIListener
 *
 * This interface defines the services requested by HubGUI.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

public interface HubGUIListener
{
    public void translate();
    public void open();
    public void close();
}

```

```

/*
 * CLASS: ImageFromFileButtonCanvas
 *
 * This class extends ImageFromFileCanvas adding a button-like
 * region in a specified part of the canvas. The button region
 * is delineated by a raised 3D rectangle which becomes inset
 * when 'pressed'.
 * @author William Valdar

```

```

 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;
import gjt.*;
import java.applet.Applet;

```

```

public class ImageFromFileButtonCanvas extends ImageFromFileCanvas
{
    private ThreeDRectangle rect;
    private Triggerable target;
    private boolean pressed = false;

    public ImageFromFileButtonCanvas(Applet applet, String filename, Color bg, Rectangle r,
Triggerable target)
    {
        super(applet, filename, bg);
        this.target = target;
        rect = new ThreeDRectangle(this, r.x, r.y, r.width, r.height);
    }

    public void paint(Graphics g)
    {
        super.paint(g);
        rect.paintRaised();
    }

    public boolean mouseDown(Event e, int x, int y)
    {
        if(rect.inside(x,y))
        {
            rect.paintInset();
            pressed = true;
            return true;
        }
        return false;
    }

    public boolean mouseUp(Event e, int x, int y)
    {
        if(pressed)

```

```

        {
            rect.paintRaised();
            pressed = false;
            target.go();
            return true;
        }
        return false;
    }
}

/*
 * CLASS: ImageFromFileCanvas
 *
 * This class represents a canvas that loads and displays a
 * specified image from file. This class could be improved
 * by providing an additional constructor(s) that accepts a
 * preloaded images.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.awt.*;
import java.awt.image.*;
import java.applet.Applet;

public class ImageFromFileCanvas extends Canvas
{
    private Image image;
    private Applet applet;
    private String filename;

    public ImageFromFileCanvas(Applet applet, String filename, Color bg)
    {
        this.applet = applet;
        this.filename = filename;
        getPicture();
        this.setBackground(bg);
    }

    public ImageFromFileCanvas(Applet applet, String filename)
    {
        this.applet = applet;
        this.filename = filename;
        getPicture();
    }

    protected void getPicture()
    {
        MediaTracker tracker = new MediaTracker(this);

        image = applet.getImage(applet.getCodeBase(), filename);

        tracker.addImage(image, 0);
        try {
            tracker.waitForID(0);
        }
        catch (InterruptedException e) {e.printStackTrace();}

        System.out.println("w=" + image.getWidth(this) + ", h=" + image.getHeight(this));
    }

    public void paint(Graphics g)
    {
        g.drawImage(image, 0, 0, this);
    }

    public void update(Graphics g)
    {
        paint(g);
    }

    public Dimension preferredSize()
    {
        return new Dimension (image.getWidth(this), image.getHeight(this));
    }
}

/*
 * CLASS: InfoDialog

```

```

*
* This class encapsulates a GUI information box that displays
* a message and is dismissed when the user clicks on its
* 'okay' button. This class used to extend Dialog to provide
* modal functionality. However, at the time of writing,
* Dialog's modal function caused screen display problems.
* InfoDialog was therefore changed to extend Frame.
* @author William Valdar
* @version 1.0 (September 1997)
*/

```

```

import java.awt.*;
import javax.swing.*;

```

```

public class InfoDialog extends Frame
{
    protected Button button;
    protected MultiLineLabel label;
    protected String message;
    protected Frame myparent;

    public InfoDialog(Frame myparent, String mytitle, String message, boolean modal)
    {
        super(mytitle);
        this.myparent = myparent;
        this.message = message;
        instantiateComponents();
        layoutComponents();
        this.show();
    }

    private void instantiateComponents()
    {
        label = new MultiLineLabel(message, 20, 20);
        button = new Button("Okay");
    }

    private void layoutComponents()
    {
        Panel p1 = new Panel();
        p1.setLayout(new FlowLayout(FlowLayout.CENTER));
        p1.add(button);

```

```

        Panel p2 = new Panel();
        p2.setLayout(new GridLayout(2,1));
        p2.add(new Separator());
        p2.add(p1);

        this.setLayout(new BorderLayout(15,15));

        this.add("Center", label);
        this.add("South", p2);
        this.pack();

        Rectangle r = myparent.getBounds();
        move( (r.x+r.width/2-this.size().width/2), (r.y+r.height/2-this.size().height/2));
    }

    public boolean action(Event event, Object what)
    {
        if(event.target == button)
        {
            die();
            return true;
        }
        else return false;
    }

    public boolean gotFocus(Event event, Object what)
    {
        button.requestFocus();
        return true;
    }

    public void die()
    {
        this.hide();
        this.dispose();
    }
}

/*
 * CLASS: InputFormDialog

```

```

*
* This class extends FrameOkayCancelDialog. It overrides the
* superclass okay() method, requesting its listener to cancel
* any FeatureInputDialog objects currently in use.
* @author William Valdar
* @version 1.0 (September 1997)
*/

import java.awt.*;
import javax.swing.*;

public class InputFormDialog extends FrameOkayCancelDialog
{
    InputFormDialogListener listener = null;

    public InputFormDialog(Locker locker, boolean modal, String subtitle, String message,
        InputFormDialogListener l)
    {
        super(locker, modal, subtitle, message);

        this.listener = l;
        this.show();
    }

    public void okay()
    {
        listener.cancelCurrentForm();
    }
}

/*
* INTERFACE: InputFormDialogListener
*
* This interface defines the services requested by
* InputFormDialog. It is currently implemented by Hub.
* @author William Valdar
* @version 1.0 (September 1997)
*/

```

```

public interface InputFormDialogListener
{
    public void cancelCurrentForm();
}

/*
* CLASS: InputFrame
*
* This class encapsulates a pop-up GUI box containing a text area.
* When the 'accept' button is pressed, whatever text has been
* entered is sent back to the MyIO parent.
* @author William Valdar
* @version 1.0 (September 1997)
*/

import java.awt.*;
import javax.swing.*;

public class InputFrame extends Frame
{
    MyIO ioObject;

    TextArea inta;
    Button bu_accept;
    Button bu_clear;
    Button bu_cancel;

    public InputFrame(MyIO ioObject)
    {
        super("Database Input");
        this.ioObject = ioObject;
        inta = new TextArea(20, 50);
        inta.setEditable(true);

        bu_accept = new Button("Accept");
        bu_clear = new Button("Clear");
        bu_cancel = new Button("Cancel");

        layoutComponents();
        this.pack();
        this.show();
    }
}

```

```

}
private void layoutComponents()
{
    Panel bp = layoutButtonPanel();

    this.add("Center", inta);
    this.add("South", bp);
}

private Panel layoutButtonPanel()
{
    Panel p1 = new Panel();
    p1.setLayout(new GridLayout(1,3));

    p1.add(bu_accept);
    p1.add(bu_clear);
    p1.add(bu_cancel);

    Separator sep = new Separator();

    Panel p2 = new Panel();
    p2.setLayout(new GridLayout(2,1));
    p2.add(sep);
    p2.add(p1);

    return p2;
}

protected void sendToIOObject(String s)
{
    ioObject.setInputText( s );
}

public boolean action(Event e, Object arg)
{
    if(e.target == bu_accept)
    {
        String s = inta.getText();
        sendToIOObject(s);
        this.hide();
        this.dispose();
        return true;
    }
    if(e.target == bu_clear)
    {
        inta.setText("");
        return true;
    }
    if(e.target == bu_cancel)
    {
        this.hide();
        this.dispose();
        return true;
    }
    return false;
}

}

/*
 * CLASS: JFeatures
 *
 * This class extends Applet and spawns the main program from
 * a Web page. It implements the SingleChildRestriction
 * interface to ensure that there is a maximum of one instance
 * of Hub and its large parent-child tree. If a second Hub was
 * instantiated, it would disastrously modify the same
 * FTED_ResourceRegistry object that the first instance of Hub
 * was using. JFeatures also implements the Triggerable
 * interface, responding to a single triggering method call,
 * go().
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.applet.*;
import java.awt.*;
import javax.swing.*;

public class JFeatures extends Applet implements Triggerable, SingleChildRestriction {

    Canvas canvas = null;
    Object child = null;

```

```

/**
 * Initialize this object.
 */
public void init()
{
    super.init();
    instantiateComponents();
    layoutComponents();
}

public void instantiateComponents()
{
    canvas = new ImageFromFileButtonCanvas(this, "start.gif", Color.white, new
Rectangle(104,106,293,69), this);
}

public void layoutComponents()
{
    this.setLayout(new BorderLayout());
    this.add("Center", canvas);
}

public void go()
{
    smartInstance();
}

//implement SingleChildRestriction

public void setChildToNull()
{
    child = null;
}

private void smartInstance()
{
    if(child == null)
    {
        child = new Hub(this, this);
    }
}

```

```

}

/**
 * The main method for running this class as a standalone application
 * @param args The argument list sent to the program.
 */
public static void main (String args[]) {
    JFeatures app = new JFeatures();
    app.init();
    app.start();
}

}

/**
 * INTERFACE: Locker
 *
 * This interface defines methods necessary to implement a simple
 * program lock, that prevents the program from performing a given
 * set of activities until "unlock()" has been called. This is
 * implemented in Hub as a simple flag. The mechanism suggested by
 * this interface is not as complicated or effective as more
 * commonly used mutex-type locks and should be modified or removed
 * in later versions.
 *
 * The Locker interface and its implementation was written to support
 * pseudo-modal behavior of the FrameOkayCancelDialog class. This
 * was felt appropriate at the time since Dialog's modal function
 * caused display problems on some platforms.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

public interface Locker
{
    public void lock();
    public void unlock();
}

```

```

/*
 * CLASS: MultiLineLabel
 *
 * This class encapsulates a GUI label that can display multiple
 * lines of text. This is an exact copy of the MultiLineLabel
 * defined in "Java in a Nutshell" by David Flanagan, O'Reilly
 * & Associates, first edition.
 * @author William Valdar (after David Flanagan)
 * @version 1.0 (September 1997)
 */

import java.awt.*;
import java.util.*;

public class MultiLineLabel extends Canvas
{
    public static final int LEFT = 0;
    public static final int CENTER = 1;
    public static final int RIGHT = 2;
    protected String[] lines;
    protected int num_lines;
    protected int margin_width;
    protected int margin_height;
    protected int line_height;
    protected int line_ascent;
    protected int[] line_widths;
    protected int max_width;
    protected int alignment = LEFT;

    protected void newLabel(String label)
    {
        StringTokenizer st = new StringTokenizer(label, "\n");
        num_lines = st.countTokens();
        lines = new String[num_lines];
        line_widths = new int[num_lines];
        for(int i=0; i<num_lines; i++) lines[i] = st.nextToken();
    }

    protected void measure()
    {
        FontMetrics fm = this.getFontMetrics(this.getFont());
        if(fm == null) return;

```

```

        line_height = fm.getHeight();
        line_ascent = fm.getAscent();
        max_width = 0;
        for(int i = 0; i < num_lines; i++)
        {
            line_widths[i] = fm.stringWidth(lines[i]);
            if(line_widths[i] > max_width) max_width = line_widths[i];
        }
    }

    public MultiLineLabel(String label, int margin_width, int margin_height, int alignment)
    {
        newLabel(label);
        this.margin_width = margin_width;
        this.margin_height = margin_height;
        this.alignment = alignment;
    }

    public MultiLineLabel(String label, int margin_width, int margin_height)
    {
        this(label, margin_width, margin_height, LEFT);
    }

    public MultiLineLabel(String label, int alignment)
    {
        this(label, 10, 10, alignment);
    }

    public MultiLineLabel(String label)
    {
        this(label, 10, 10, LEFT);
    }

    public void setLabel(String label)
    {
        newLabel(label);
        measure();
        repaint();
    }

    public void setFont(Font f)
    {

```

```

        super.setFont(f);
        measure();
        repaint();
    }

    public void setForeground(Color c)
    {
        super.setForeground(c);
        repaint();
    }

    public void setAlignment(int a){ this.alignment = a;}
    public void setMarginWidth(int mw){ this.margin_width = mw;}
    public void setMarginHeight(int mh){this.margin_height = mh;}
    public int getAlignment(){return alignment;}
    public int getMarginWidth(){return margin_width;}
    public int getMarginHeight(){return margin_height;}

    public void addNotify(){super.addNotify(); measure();}

    public Dimension preferredSize()
    {
        return new Dimension(max_width +2*margin_width,
                               num_lines*line_height +2*margin_height);
    }

    public Dimension minimumSize()
    {
        return new Dimension(max_width, num_lines*line_height);
    }

    public void paint(Graphics g)
    {
        int x, y;
        Dimension d = this.size();
        y = line_ascent + (d.height - num_lines*line_height)/2;
        for(int i=0; i<num_lines; i++, y += line_height)
        {
            switch(alignment){
                case LEFT:
                    x = margin_width; break;
                case CENTER:
                    x = d.width - margin_width - line_widths[i];
                    break;
                case RIGHT:
                    x = d.width - margin_width - line_widths[i];
                    break;
            }
            g.drawString(lines[i], x,y);
        }
    }
}

```

```

        x = (d.width - line_widths[i])/2; break;
    case RIGHT:
        x = d.width - margin_width - line_widths[i];
        break;
    }
    g.drawString(lines[i], x,y);
}
}
}

/*
 * CLASS: MyDrawnShape
 *
 * This class encapsulates a drawn shape that may be rectangular,
 * oval, or egg-timer-shaped. Objects of this class are used
 * to represent features in both the DiagramOfFeatures and in the
 * GraphicFeatureTextCanvas classes.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import gjt.*;
import java.awt.*;

public class MyDrawnShape
{
    final static int RECTANGLE = 0;
    final static int OVAL = 1;
    final static int EGGTIMER = 2;

    private Polygon eggpoly;
    private DrawnRectangle drect;
    private int myShape;
    private boolean visible;
    private Component drawInto;

    public MyDrawnShape(Component drawInto, int thick, int shape,
                        int x, int y, int w, int h,

```

```

        boolean visible)
    {
        this.myShape = shape;
        this.drawInto = drawInto;
        this.visible = visible;
        direct = new DrawnRectangle(drawInto, thick, x, y, w, h);

        eggpoly = new Polygon();
        reshapeEggtimer(x,y,w,h);
    }

    public MyDrawnShape(Component drawInto, int thick,
                        int x, int y, int w, int h)
    {
        this(drawInto, thick, RECTANGLE, x,y,w,h,true);
    }

    public MyDrawnShape(Component drawInto)
    {
        this(drawInto, 1, RECTANGLE, 0,0,0,0, true);
    }

    public void setLineColor(Color linecolor)
    {
        direct.setLineColor(linecolor);
    }

    public Color getLineColor()
    {
        return direct.getLineColor();
    }

    public void setFillColor(Color fillcolor)
    {
        direct.setFillColor(fillcolor);
    }

    public Color getFillColor()
    {
        return direct.getFillColor();
    }

```

```

    public void setShape(int shape)
    {
        if(shape == RECTANGLE || shape == OVAL || shape == EGGTIMER)
        {
            myShape = shape;
        }
    }

    public int getShape()
    {
        return myShape;
    }

    public void setVisible(boolean visible)
    {
        this.visible = visible;
    }

    public boolean isVisible()
    {
        return visible;
    }

    public void reshape(int x, int y, int w, int h)
    {
        direct.reshape(x,y,w,h);
        reshapeEggtimer(x,y,w,h);
    }

    public void reshapeEggtimer(int x, int y, int w, int h)
    {
        h--;
        int[] xpoints = {x,x+w,x,x+w,x};
        int[] ypoints = {y,y,y+h,y+h,y};

        eggpoly.xpoints = xpoints;
        eggpoly.ypoints = ypoints;
        eggpoly.npoints = 5;
    }

    public boolean inside(int x, int y)
    {
        switch(myShape)

```

```

        {
            case RECTANGLE:
                return drect.inside(x,y);
            case OVAL:
                return drect.inside(x,y);
            case EGGTIMER:
                return eggpoly.inside(x,y);
        }
        return false;
    }

    public void paint()
    {
        if(visible)
        {
            switch(myShape)
            {
                case RECTANGLE:
                    paintRectangle();
                    break;
                case OVAL:
                    paintOval();
                    break;
                case EGGTIMER:
                    paintEggtimer();
                    break;
            }
        }
    }

    public void paintRectangle()
    {
        drect.fill();
        drect.paint();
    }

    public void paintOval()
    {
        Graphics g = drawInto.getGraphics();

        int thick = drect.getThickness();

        int x = drect.x;
        int y = drect.y;
        int w = drect.width;
        int h = drect.height;

        //Do fill
        g.setColor(drect.getFillColor());
        g.fillOval(x, y, w, h);

        //Do lines
        g.setColor(drect.getLineColor());

        for(int i = 0; i<thick; i++)
        {
            g.drawOval(x+i, y+i, w-(2*i), h-(2*i));
        }
    }

    public void paintEggtimer()
    {
        Graphics g = drawInto.getGraphics();

        g.setColor(drect.getFillColor());
        g.fillPolygon(eggpoly);

        g.setColor(drect.getLineColor());
        g.drawPolygon(eggpoly);
    }
}

/*
 * CLASS: MyIO
 *
 * This class controls the I/O operations of J-Features,
 * coordinating user input and output with conversions between
 * internal and database formats. It acts as a simple layer
 * between Hub and the InputFrame and ObjectFlatFileConverter
 * classes.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

*/

import java.awt.*;
import java.util.*;

public class MyIO
{
    ObjectFlatFileConverter t;
    MyIOListener myListener;

    String dummySequence;

    Frame inputframe = null;

    public MyIO(MyIOListener myListener, ObjectFlatFileConverter t)
    {
        this.myListener = myListener;
        this.t = t;

        int dummylength = 2000;
        char[] d = new char[dummylength];
        for(int i=0; i<dummylength; i++)
        {
            d[i] = 'X';
        }
        dummySequence = new String(d);
    }

    public void give(DBSeqInfo dbsi)
    {
        Vector v = t.translateToFlat(dbsi);

        String s = MyUtils.VectorOfStringsToString( v );

        MyViewer mv = new MyViewer(s, "Database Format", "Cut and paste into
another text editor",
                                20, 70, new Font("Courier",
Font.PLAIN, 12) );
    }

    public void receive()

```

```

    {
        inputframe = new InputFrame(this);
    }

    public void setInputText(String s)
    {
        Vector v = MyUtils.StringToVector(s);
        DBSeqInfo data = null;
        try
        {
            data = t.translateToOb(v);
        }
        catch(Exception e) {e.printStackTrace();}
        if(data != null)
        {
            if(data.hasSequence() == false)
            {
                data.setSequence(dummySequence);
            }
            myListener.setDBSeqInfo(data);
        }
        inputframe = null;
    }
}

/*
 * INTERFACE: MyIOListener
 *
 * This interface defines the services requested by the
 * MyIO object. It is currently implemented by Hub.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

public interface MyIOListener
{
    public void setDBSeqInfo(DBSeqInfo dbsi);
}

/*

```

```
* CLASS: MyUtils
```

```
*
```

```
* This class contains a collection of convenience methods
* used throughout the program. If a method was needed in
* a number of places it was put in this class. One goal of
* this class was to reduce overall coupling of the program
* with the jgl package, the sorting methods of which are
* used extensively.
```

```
* @author William Valdar
```

```
* @version 1.0 (September 1997)
```

```
*/
```

```
import java.util.*;
import COM.objectspace.jgl.*;
import java.awt.*;
```

```
public class MyUtils extends Object
{
```

```
    //Vectors take in null lines calling them "null". This gets rid of "null" lines.
```

```
    public static Vector ridVectorOfEmptyStrings (Vector inv)
    {
        Vector outv = new Vector(); Object obj;

        int invsize = inv.size();

        for(int i = 0; i<invsize; i++)
        {
            obj = inv.elementAt(i);

            if( obj.toString().length() != 0 )
            {
                outv.addElement( obj );
            }
        }
        return outv;
    }
}
```

```
public static Vector StringToVector (String string)
{
```

```
    Vector vector = new Vector();
```

```
    StringTokenizer st = new StringTokenizer( string , "\n" );
```

```
    while(st.hasMoreElements())
    {
        vector.addElement( st.nextToken() );
    }
}
```

```
    return vector;
```

```
public static String VectorOfStringsToString( Vector vector)
{
```

```
    String string = new String();
```

```
    int vsize = vector.size();
```

```
    for(int i = 0; i<vsize; i++)
```

```
    {
        string += (String) vector.elementAt(i) + "\n";
    }
}
```

```
    return string;
```

```
public static Vector getVectorOfEnumeration( Enumeration e )
{
```

```
    Vector vector = new Vector();
```

```
    while(e.hasMoreElements())
```

```
    {
        vector.addElement(e.nextElement());
    }
}
```

```
    return vector;
```

```
public static void sortVectorOfStringsFwd( Vector vector )
{
```

```
    sortVector( vector, new LessEqualString() );
```

```

    }

    public static void sortVector( Vector vector, BinaryPredicate comparator )
    {
        VectorArray va = new VectorArray( vector );

        Sorting.sort( va , comparator );
    }

    //AWT helper

    public static Color whiter(Color c)
    {
        return new Color( (c.getRed() + 255 )/2,
                          (c.getGreen() + 255)/2,
                          (c.getBlue() + 255)/2 );
    }

    public static void choiceItemsFromStringVector(Choice choice, Vector stringvector)
    {
        int vsize = stringvector.size();

        for(int i = 0; i<vsize; i++)
        {
            choice.addItem((String)stringvector.elementAt(i));
        }
    }
}

/*
 * CLASS: MyViewer
 *
 * This class encapsulates a pop -up GUI information box containing
 * a text window. It allows the parent to specify the dimensions of
 * the text area and the font of the data. MyViewer is used to
 * display output in database format by the MyIO class.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;
import gjt.*;

public class MyViewer extends Frame
{
    Button bu_close;
    TextArea textarea;
    String subtitle;
    int rows, columns;
    Font f = null;
    String data;

    public MyViewer(String data, String mytitle, String subtitle, int rows, int columns)
    {
        super(mytitle);
        this.subtitle = subtitle;
        this.rows = rows;
        this.columns = columns;
        this.data = data;

        doAWTStuff();
    }

    public MyViewer(String data, String mytitle, String subtitle, int rows, int columns, Font f)
    {
        super(mytitle);
        this.subtitle = subtitle;
        this.data = data;
        this.rows = rows;
        this.columns = columns;
        this.f = f;

        doAWTStuff();
    }

    public MyViewer(String data, int rows, int columns)
    {
        this(data, "MyViewer", "Viewer", rows, columns);
    }

    public MyViewer(String data)
    {

```

```

        this(data, 20, 40);
    }

    public MyViewer(String data, String title, String subtitle)
    {
        this(data, title, subtitle, 20, 40);
    }

    private void doAWTStuff()
    {
        instantiateComponents();
        initializeComponents();
        layoutComponents();
        this.pack();
        this.show();
    }

    private void instantiateComponents()
    {
        bu_close = new Button("Close");

        textarea = new TextArea(new String(data), rows, columns);
    }

    private void initializeComponents()
    {
        textarea.setEditable(false);
        if(f != null)
        {
            textarea.setFont(f);
        }
    }

    private void layoutComponents()
    {
        Panel bp = layoutButtonPanel();
        Panel contain_all = new Panel();

        contain_all.setLayout(new BorderLayout());
        contain_all.add("Center", textarea);
        contain_all.add("South", bp);

        Box b1 = new Box(contain_all, subtitle);

```

```

        this.add("Center", b1);
    }

    private Panel layoutButtonPanel()
    {
        Separator sep = new Separator();

        Panel p1 = new Panel();
        p1.setLayout(new GridLayout(2,1));

        p1.add(sep);
        p1.add(bu_close);

        return p1;
    }

    public boolean action(Event e, Object what)
    {
        if(e.target == bu_close)
        {
            this.hide();
            this.dispose();
            return true;
        }
        return false;
    }
}

/*
 * INTERFACE: ObjectFlatFileConverter
 *
 * This interface defines the methods necessary for converting
 * between flat file format (as a vector of strings, each of
 * represents a line) and object format (DBSeqInfo).
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.util.*;

```

```

public interface ObjectFlatFileConverter
{
    public Vector translateToFlat(DBSeqInfo dbsi);
    public DBSeqInfo translateToOb(Vector stringvector) throws NoSuchFieldError,
Exception;
}

```

```

/*
 * CLASS: SequenceColors
 *
 * This class defines the methods necessary to initialize
 * data stores containing information about allowed colours,
 * colour names, and default colour/letter associations for
 * the DiagramOfSequence. Like FeatureColorsROR this might
 * be more appropriately placed in initialization procedures
 * within the DiagramOfSequence itself.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.util.*;
import java.awt.*;

```

```

public class SequenceColors
{
    private Hashtable residues, bases;
    private TwoWayHashtable availableColors;
    private Vector vresidues, vbases, availableColorNames;

    Color GRAY = Color.lightGray;
    Color RED = Color.red.darker();
    Color BLUE = MyUtils.whiter(Color.blue);
    Color YELLOW = Color.yellow;
    Color ORANGE = Color.orange;
    Color CYAN = Color.cyan.darker();
    Color MAGENTA = Color.magenta;
    Color PINK = Color.pink.darker();
    Color GREEN = Color.green;

    public SequenceColors()
    {

```

```

        initialize();
    }

    private void initialize()
    {
        initializeResidues();
        initializeBases();
        initializeAvailableColors();
    }

    private void initializeResidues()
    {
        Hashtable r = new Hashtable();
        Vector v = new Vector();

        char[] nonpolar = { 'A', 'G', 'T', 'L', 'V' };
        char[] acidic = { 'D', 'E' };
        char[] basic = { 'H', 'K', 'M' };
        char[] sulphurous = { 'C', 'M' };
        char[] aliphatic = { 'S', 'T' };
        char[] aromatic = { 'F', 'W', 'Y' };
        char proline = 'P';

        String s = null;

        for(int i = 0; i<nonpolar.length; i++)
        {
            s = String.valueOf(nonpolar[i]);
            r.put( s,GRAY);
            v.addElement(s);
        }
        for(int i = 0; i<acidic.length; i++)
        {
            s = String.valueOf(acidic[i]);
            r.put( s,RED);
            v.addElement(s);
        }
        for(int i = 0; i<basic.length; i++)
        {
            s = String.valueOf(basic[i]);
            r.put( s,BLUE);
            v.addElement(s);
        }
    }

```

```

for(int i = 0; i<sulphurous.length; i++)
{
    s = String.valueOf(sulphurous[i]);
    r.put( s, YELLOW );
    v.addElement(s);
}
for(int i = 0; i<aliphatic.length; i++)
{
    s = String.valueOf(aliphatic[i]);
    r.put( s, CYAN );
    v.addElement(s);
}
for(int i = 0; i<aromatic.length; i++)
{
    s = String.valueOf(aromatic[i]);
    r.put( s, GREEN );
    v.addElement(s);
}

s = String.valueOf(proline);
r.put( s, PINK );
v.addElement(s);

s = String.valueOf('B');
r.put( s, ORANGE );
v.addElement(s);

s = String.valueOf('Z');
r.put( s, ORANGE );
v.addElement(s);

s = String.valueOf('X');
r.put( s, GRAY );
v.addElement(s);

s = "default";
r.put( s, GRAY );
v.addElement(s);

MyUtils.sortVectorOfStringsFwd(v);

residues = r;
vresidues = v;
}

private void initializeBases()
{
    bases = new Hashtable();
    vbases = new Vector();

    String s = null;

    s = String.valueOf('A');
    bases.put( s, RED );
    vbases.addElement(s);

    s = String.valueOf('C');
    bases.put( s, YELLOW );
    vbases.addElement(s);

    s = String.valueOf('G');
    bases.put( s, GREEN );
    vbases.addElement(s);

    s = String.valueOf('T');
    bases.put( s, BLUE );
    vbases.addElement(s);

    s = String.valueOf('U');
    bases.put( s, BLUE );
    vbases.addElement(s);

    s = "default";
    bases.put( s, GRAY );
    vbases.addElement(s);
}

private void initializeAvailableColors()
{
    availableColors = new TwoWayHashtable();

    availableColors.put("black", Color.black);
    availableColors.put("blue", Color.blue);
    availableColors.put("cyan", Color.cyan);
    availableColors.put("dark gray", Color.darkGray);
    availableColors.put("gray", Color.gray);
}

```

```

        availableColors.put("green", Color.green);
        availableColors.put("light gray", Color.lightGray);
        availableColors.put("magenta", Color.magenta);
        availableColors.put("orange", Color.orange);
        availableColors.put("pink", Color.pink);
        availableColors.put("red", Color.red);
        availableColors.put("white", Color.white);
        availableColors.put("yellow", Color.yellow);
        availableColors.put("dark red", RED);
        availableColors.put("light blue", BLUE);
        availableColors.put("dark cyan", CYAN);
        availableColors.put("dark pink", PINK);

        availableColorNames =
MyUtils.getVectorOfEnumeration(availableColors.keys());
        MyUtils.sortVectorOfStringsFwd(availableColorNames);
    }

    public Color getResidueColor(char c)
    {
        return getResidueColor( String.valueOf(c));
    }

    public Color getResidueColor(String res)
    {
        res = res.toUpperCase();
        if(residues.containsKey(res))
        {
            return (Color)residues.get(res);
        }
        return (Color)residues.get("default");
    }

    public Color getBaseColor(char c)
    {
        return getBaseColor(String.valueOf(c));
    }

    public Color getBaseColor(String base)
    {
        base = base.toUpperCase();
        if(bases.containsKey(base))
        {
            return (Color)bases.get(base);
        }
        return (Color)bases.get("default");
    }
}

public boolean replaceResidueColor(String ch, Color c)
{
    if(residues.containsKey(ch))
    {
        residues.remove(ch);
        residues.put(ch,c);
        return true;
    }
    else return false;
}

public boolean replaceBaseColor(String ch, Color c)
{
    if(bases.containsKey(ch))
    {
        bases.remove(ch);
        bases.put(ch,c);
        return true;
    }
    return false;
}

public TwoWayHashtable getAvailableColors()
{
    return availableColors;
}

public Vector getAvailableResidues()
{
    return vresidues;
}

public Vector getAvailableBases()
{
    return vbases;
}

public Vector getAvailableColorNames()

```

```

    {
        return availableColorNames;
    }
}

```

```

/*
 * INTERFACE: SingleChildRestriction
 *
 * This interface defines the methods necessary to ensure
 * that the implementing class can only contain a single
 * child of a certain type. The actual methods it defines
 * are not particularly important. Its main purpose is to
 * allow its implementing class to be recognized by other
 * classes in this way. Hub only accepts a parent that
 * implements this interface, since Hub must not have
 * multiple instances.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

interface SingleChildRestriction
{
    public void setChildToNull();
}

```

```

/*
 * CLASS: SwissDetails
 *
 * This class is a non-abstract subclass of DBDetails that
 * contains information about the standard feature keys/types
 * for a particular database. This may not be the most effective
 * or appropriate way to store additional information about a
 * particular database.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.util.*;
import java.awt.*;

```

```

import COM.objectspace.jgl.*;

public class SwissDetails extends DBDetails
{
    public Vector std_types;

    public SwissDetails()
    {
        initializeStandardTypes();
    }

    public void initializeStandardTypes()
    {
        std_types = new Vector();
        std_types.addElement("CONFLICT");
        std_types.addElement("VARIANT");
        std_types.addElement("VARSPLOC");
        std_types.addElement("MUTAGEN");
        std_types.addElement("MOD_RES");
        std_types.addElement("LIPID");
        std_types.addElement("DISULFID");
        std_types.addElement("CARBOHYD");
        std_types.addElement("METAL");
        std_types.addElement("BINDING");
        std_types.addElement("TRANSIT");
        std_types.addElement("PROPEP");
        std_types.addElement("CHAIN");
        std_types.addElement("PEPTIDE");
        std_types.addElement("DOMAIN");
        std_types.addElement("NP_BIND");
        std_types.addElement("ZN_FING");
        std_types.addElement("SIMILAR");
        std_types.addElement("REPEAT");
        std_types.addElement("HELIX");
        std_types.addElement("TURN");
        std_types.addElement("STRAND");
        std_types.addElement("ACT_SITE");
        std_types.addElement("SITE");
        std_types.addElement("INIT_MET");
        std_types.addElement("NON_TER");
        std_types.addElement("NON_CONS");
        std_types.addElement("SIGNAL");
    }
}

```

```

        MyUtils.sortVectorOfStringsFwd( std_types );
    }

    public Vector getStandardTypes()
    {
        return std_types;
    }
}

/*
 * CLASS: SwissFTEntry
 *
 * This class is a non-abstract subclass of FeatureTableEntry that
 * is specific for the SwissProt database. J-Features would be more
 * flexible if this was removed in favour of a robust non-abstract
 * version of FeatureTableEntry.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.util.*;
import java.awt.*;

public class SwissFTEntry extends FeatureTableEntry
{
    protected static final String format = "SWISSPROT";
    protected static String DEFAULT_TYPE = "[NONE]";
    protected Vector qualifiers;
    protected Vector comments;

    public SwissFTEntry()
    {
        this(new Point(0,0));
    }

    public SwissFTEntry(Point loc)
    {

```

```

        setLocation(loc);
        setType(DEFAULT_TYPE);
    }

    public String getFormat(){return format;}

    public void setLocation(Point location){this.location=location;}
    public void setType(String type){this.type=type;}

    public Point getLocation(){return this.location;}
    public String getType(){return this.type;}

    public String getDEFAULT_TYPE(){return this.DEFAULT_TYPE;}

    public void setQualifiers(Vector qualifiers){this.qualifiers=qualifiers;}
    public void setComments(Vector comments){this.comments=comments;}

    public boolean qualifiersExist()
    {
        if(qualifiers != null)
        {
            if(qualifiers.isEmpty() == false)
            {
                return true;
            }
        }
        return false;
    }

    public boolean commentsExist()
    {
        if(comments != null)
        {
            if(comments.isEmpty() == false)
            {
                return true;
            }
        }
        return false;
    }

    public Vector getQualifiers(){return qualifiers;}
    public Vector getComments(){return comments;}

```

```

public boolean isRegion()
{
    if(location.x == location.y) return false;
    else return true;
}

//needs to be changed

public boolean equals(Object obj)
{
    /*      if(obj instanceof SwissFTEntry)
    {
        SwissFTEntry sfte = (SwissFTEntry)obj;
        return (this.location.equals(sfte.location)) &&
(this.type.equals(sfte.type));
    }
    */      return false;
}

public String toString()
{
    String s=new String ();
    s += "Type: " + type + "\n";
    s += "From: " + location.x + "\n";
    s += "To: " + location.y + "\n";
    if(qualifiers!=null)
    {
        int qsize = qualifiers.size();
        s += qsize + " qualifiers\n";
        for(int i=0; i<qsize; i++)
        {
            s += "Qualifier (" + (i+1) + "): " + (String)
qualifiers.elementAt(i) + "\n";
        }
    }
    if(comments!=null)
    {
        int csize = comments.size();
        s += csize + " comments\n";
        for(int i=0; i<csize; i++)
        {

```

```

            s += "Comment (" + (i+1) + "): " + (String)
comments.elementAt(i) + "\n";
        }
    }
    return s;
}

public boolean isUpstreamOf(FeatureTableEntry fte)
{
    Point other = fte.getLocation();

    if(this.location.equals(other) ) return false;

    if(this.location.x == other.x)
    {
        if(this.location.y < other.y)      return true;

        else return false;
    }
    else
    {
        if(this.location.x < other.x)      return true;

        else return false;
    }
}

public Object clone()
{
    SwissFTEntry fte = new SwissFTEntry();
    fte.location = new Point(location.x, location.y);
    fte.type = new String(type);
    if(qualifiersExist())
    {
        fte.qualifiers = (Vector) qualifiers.clone();
    }
    if(commentsExist())
    {
        fte.comments = (Vector) comments.clone();
    }
    return fte;
}

```

```

}
/*
 * SwissInfo is an object for storing information about entries
 * from the SwissProt database. It is supposed to implement the
 * abstract methods of its superclass in a format dependant way.
 * Although it is not obvious how another database format might
 * implement these methods differently, some formats may require
 * additional methods or data members as they are being translated
 * to and from db format (See SwissProtTranslator and
 * ObjectToFlatConverter). In fact, getOriginalInfo() could well
 * be an example of a field used during translation only, i.e.
 * when the object is treated as the derived class rather than
 * as DBSeqInfo.
 * @author William Valdar
 * @version 1.0
 */

import java.util.*;

public class SwissInfo extends DBSeqInfo
{
    /*
     *Protected members describing the main features of a
     *database entry
     */

    protected static String format = "SWISSPROT";
    protected String accession = null;
    protected String id = null;
    protected String sequence = null;
    protected Vector features = null;
    protected Vector original_info = null;

    /*
     *Default values to help avoid null pointer exceptions
     *from incompletely initialized objects
     */

    protected String NOSEQUENCE = "NO SEQUENCE";
    protected String NOACCESSION = "NO ACCESSION";

    /*
     *Simple get and set functions

```

```

 */

    public synchronized String getFormat(){return format;}
    public synchronized String getAccession(){return accession;}
    public synchronized String getId(){return id;}
    public synchronized String getSequence(){return sequence;}
    public synchronized Vector getFeatures(){return features;}
    public synchronized Vector getOriginalInfo(){return original_info;}

    public synchronized void setAccession(String accession){ this.accession = accession; }
    public synchronized void setId(String id){ this.id = id; }
    public synchronized void setSequence(String sequence){ this.sequence = sequence;}
    public synchronized void setFeatures(Vector features){ this.features = features; }
    public synchronized void setOriginalInfo(Vector original_info){ this.original_info =
original_info; }

    /*
     *Boolean 'has got a' functions
     */

    public boolean hasSequence()
    {
        if(sequence != null && !sequence.equals("null") &&
!sequence.toString().equals("null"))
        {
            if(sequence.length()> 0);
            {
                if(sequence != NOSEQUENCE)
                {
                    return true;
                }
            }
        }
        return false;
    }
    public synchronized boolean hasFeatures(){ return features != null; }
    public synchronized boolean hasOriginalInfo(){ return original_info != null; }

    /*
     *Constructor initialized with defaults
     */

    public SwissInfo()

```

```

{
    accession = NOACCESSION;

    id = "NO ID";

    sequence = NOSEQUENCE;

    features = null;

    original_info = null;
}

/*
 * Overrides Object.toString()
 */
public String toString()
{
    String string = new String("Format: "+format+"\n");

    string += "Id : " + id + "\n";

    string += "Accession Number : " + accession + "\n";

    string += "Sequence : " + sequence + "\n";

    if( features != null ) string += features.toString();

    else string += "No Features\n";

    return string;
}

/*
 * Overrides Object.clone()
 */
public synchronized Object clone()
{
    SwissInfo si = new SwissInfo();
    if(accession != null)
    {
        si.accession = new String(accession);
    }
}

```

```

}
if(id != null)
{
    si.id = new String(id);
}
if(sequence != null)
{
    si.sequence = new String(sequence);
}
if(features != null)
{
    si.features = (Vector) features.clone();
}
if(original_info != null)
{
    si.original_info = (Vector) original_info.clone();
}

return si;
}

}

/*
 * CLASS: SwissProtTranslator
 *
 * This class implements the ObjectFlatFileConverter interface and
 * contains the methods necessary for parsing SwissProt documents
 * and extracting information about the feature table and sequence;
 * saving extracted information about the feature table and the
 * sequence as a SwissInfo instance of a DBSeqInfo object; sorting
 * an array of FeatureTableEntry objects; saving information from a
 * DBSeqInfo object in SwissProt format.
 *
 * This class contains a number of general string manipulation
 * methods that would be useful in a class that provided similar
 * services for a different database format. It may have been
 * useful to contain these methods either within a separate utility
 * class or within a superclass from which database specific
 * translation classes were derived.
 * @author William Valdar
 * @version 1.0 (September 1997)

```

```

*/

import java.util.*;
import COM.objectspace.jgl.*;
import java.awt.*;

public class SwissProtTranslator extends Object implements ObjectFlatFileConverter
{

//=====
//Members

    protected static final int KEY_COL = 6;
    protected static final int FROM_END_COL = 20;
    protected static final int TO_END_COL = 27;
    protected static final int DESCRIPT_COL = 35;
    protected static final String COMM_MARK ="/";

    protected static final String FT_MARK = "FT";
    protected static final String SEQ_MARK = "SQ";
    protected static final String HTML_MARK = "<HTML>";
    protected static final String ID_MARK = "ID ";
    protected static final String AC_MARK = "AC";
    protected static final String HTML_REF_MARK = "RX";

    protected static final int MARK_LENGTH = 2;
    protected static final int MAX_TYPE_LENGTH = 8;
    protected static final int MAX_FROM_LENGTH = 999999;
    protected static final int MAX_TO_LENGTH = 999999;
    protected static final int MAX_QUALIFIER_LENGTH = 40;    //". " or "," has to be
added at end
    protected static final int MAX_COMMENT_LENGTH;

    public static Vector STD_TYPES;

//=====
//Initialization

    static
    {

```

```

MAX_COMMENT_LENGTH = 41 - COMM_MARK.length();

//ResourceSwissProt r = ResourceSwissProt.instance();

//STD_TYPES = r.STD_TYPES;
}

public Vector getSTD_TYPES(){ return STD_TYPES; }

//=====
//Public interface methods

//Implements ObjectFlatFileConverter.translateToOb

public DBSeqInfo translateToOb(Vector stringvector) throws NoSuchFieldError,
Exception
{
    return extractInfo(stringvector);
}

//Implements ObjectFlatFileConverter.translateToFlat

public Vector translateToFlat( DBSeqInfo dbsi)
{
    //Could have some format checks here

    SwissInfo swissinfo = (SwissInfo) dbsi;

    return presentSwissInfo(swissinfo);
}

//Trims the fields of each feature object in the vector
//so that e.g. the qualifiers can fit into the column space available
//in SwissProt format

public void trimFeatures(Vector features)
{
    int vsize = features.size();

```

```

        for(int i = 0; i<vsize; i++)
        {
            trimAllFields((SwissFTEntry) features.elementAt(i));
        }
    }

//Puts FT Entries in order of starting position
//If two features have the same starting position,
//the one that ends first has priority

public void sortFTEntries(Vector features)
{
    VectorArray va = new VectorArray( features );

    BinaryPredicate comparator = new FeatureTableEntryPredicate();

    Sorting.sort( va, comparator);
}

//=====
//Protected methods used by more than one public interface method's subroutines

//Finds which element string in a vector a strings has the specified line mark

public int findLineIndex(String descriptLine, Vector textv)
{
    int textvsize = textv.size() - 1;

    for(int i=0; i<textvsize; i++)
    {
        if( ( (String) textv.elementAt(i) ).startsWith(descriptLine) )
        {
            return i;
        }
    }
    throw new NoSuchFieldError(descriptLine);
}

//Finds the last string in the string vector to have the specified line index

public int findLastLineIndex(String descriptLine, Vector textv)
{

```

```

    int index = textv.size() - 1;

    for(int i = index; i >= 0; i--)
    {
        if( ((String) textv.elementAt(i) ).startsWith(descriptLine) )
        {
            return i;
        }
    }
    throw new NoSuchFieldError(descriptLine);
}

//Returns false if line does not exist

public boolean lineExists(String descriptLine, Vector text)
{
    int textsize = text.size();

    for(int i = 0; i<textsize ; i++)
    {
        if( ( (String) text.elementAt(i) ).startsWith(descriptLine) ) return
true;
    }

    return false;
}

//=====
//Protected methods for translateToOb

//NB clean_info == a non-html SwissProt entry as a vector of strings

//Get ID line

protected String extractId(Vector clean_info)
{
    String id = getLine(ID_MARK, clean_info);

    return extractLineContents(id);
}

```

```

//Get AC line accession number
protected String extractAccession(Vector clean_info)
{
    String ac = getLine(AC_MARK, clean_info);

    ac = extractLineContents(ac);

    ac = chopOffLastChar(ac);

    return ac;
}

//Get sequence part of SwissProt file
protected String extractSequence(Vector clean_info)
{
    String sequence, seq_info, temps;

    int index = findLineIndex(SEQ_MARK, clean_info);

    int total_indices = clean_info.size();

    seq_info = extractLineContents( getLine(SEQ_MARK, clean_info) );

    temps = new String();

    for(int i=index + 1; i<total_indices; i++)
    {
        temps += (String) clean_info.elementAt(i);
    }

    StringTokenizer st = new StringTokenizer(temps, " ");

    sequence = new String();

    while(st.hasMoreTokens())
    {
        sequence += st.nextToken();
    }

    //get rid of "/" marks

```

```

        sequence = chopOffLastChar( chopOffLastChar( sequence ) );

        return sequence;
    }

//Gets original SwissProt entry. Makes nohtml conversion if nec, then returns the clean info
protected Vector extractOriginalInfo(Vector text)
{
    if( isHTML(text) )
    {
        text = noHTML(text);
    }

    return text;
}

//Wraps parseFT
protected Vector extractFeatures(Vector clean_info) throws NoSuchFieldError, Exception
{
    return parseFT( getLines(FT_MARK, clean_info) );
}

//Main extract method. Takes in an html-dirty or clean SwissProt entry and returns an
object
protected SwissInfo extractInfo(Vector alltext) throws NoSuchFieldError, Exception
{
    alltext = MyUtils.ridVectorOfEmptyStrings( alltext );

    Vector clean_info;

    clean_info = extractOriginalInfo( alltext );

    SwissInfo swissinfo = new SwissInfo();

    swissinfo.setOriginalInfo( clean_info );

```

```

        if( lineExists(ID_MARK, clean_info) )    swissinfo.setId( extractId(
clean_info ) );

        if( lineExists(AC_MARK, clean_info) )    swissinfo.setAccession(
extractAccession( clean_info ) );

        if( lineExists(SEQ_MARK, clean_info) )    swissinfo.setSequence(
extractSequence( clean_info ) );

        if( lineExists(FT_MARK, clean_info) )    swissinfo.setFeatures(
extractFeatures( clean_info ) );

        return swissinfo;
    }

//Determines whether or not text is an html document
public boolean isHTML(Vector v)
{
    int vsize = v.size();

    boolean is_html = false;

    for(int i=0; i<vsize; i++)
    {
        if( ( (String)v.elementAt(i) ).indexOf(HTML_MARK) != -1 )
        {
            is_html = true;
        }
    }

    return is_html;
}

//Removes any html marks except those that are links to paper references
protected Vector noHTML(Vector dirty)
{
    Vector clean = new Vector();

    String startmark = new String(ID_MARK);

    String hmark ="<";

```

```

    String temp;

    int dirtysize = dirty.size();

    int count = 0;

    do
    {
        count++;
    }
    while( ( ((String)dirty.elementAt(count)).indexOf(startmark) == -1) &&
(count<dirtysize) );

    temp = new String((String)dirty.elementAt(count));

    int index = temp.indexOf(startmark);

    temp = temp.substring(index);

    clean.addElement(temp);

    for(int i=++count; i<dirtysize; i++)
    {
        if( (((String)dirty.elementAt(i)).indexOf(hmark) == -1)
        ||
        (((String)dirty.elementAt(i)).startsWith(HTML_REF_MARK)) )
        {
            clean.addElement((String) dirty.elementAt(i));
        }
    }

    return clean;
}

//Returns the contents of a line, minus the line mark (eg. minus 'ID')
protected String extractLineContents(String line)
{
    return line.substring( MARK_LENGTH ).trim();
}

//Gets the line with the specified line mark

```

```

protected String getLine(String descriptLine, Vector swissDoc) throws NoSuchFieldError
{
    String matching_line;

    int docsize = swissDoc.size();

    boolean line_exists = false;

    for(int i=0; i<docsize; i++)
    {
        if( ((String) swissDoc.elementAt(i)).startsWith(descriptLine) )
        {
            matching_line = new String( (String)
swissDoc.elementAt(i));

            line_exists = true;

            return matching_line;
        }
        throw new NoSuchFieldError(descriptLine);
    }
}

//Gets the lines with the specified line mark
protected Vector getLines(String descriptLine, Vector swissDoc) throws NoSuchFieldError
{
    Vector matching_lines = new Vector();

    int docsize = swissDoc.size();

    boolean line_exists = false;

    for(int i=0; i<docsize; i++)
    {
        if( ((String) swissDoc.elementAt(i)).startsWith(descriptLine) )
        {
            matching_lines.addElement(swissDoc.elementAt(i));

            line_exists = true;
        }
    }
}

        if(line_exists)      return matching_lines;
        else throw new NoSuchFieldError(descriptLine);
    }
}

//-----
//Methods for parseFT

//Returns true if the argument line is a feature table line
protected boolean isFeatureLine(String line)
{
    if(line.startsWith(FT_MARK)) return true;
    else return false;
}

//Returns true if the argument line is a new feature rather than the continuation
//another
protected boolean isNewFeature(String line)
{
    if( (line.charAt( (KEY_COL-1) ) != ' ') && (isFeatureLine(line)) )
        return true;
    else return false;
}

//Puts a SwissFTEntry together given its constituent parts
public void compileEntry(SwissFTEntry sfte, Vector qualifiers, Vector comments)
{
    sfte.setQualifiers(qualifiers);
    sfte.setComments(comments);
}

//Extracts all the essential info from a new feature line
//i.e. the key (or type) and the start and end positions
protected void extractMinimumFTDetails(SwissFTEntry sfte, String line)
{
    String tempstring, dump;
    int from, to;

```

```

StringTokenizer st = new StringTokenizer(line, " ");

dump = st.nextToken(); //remove FT bit

tempstring = st.nextToken();

sfte.setType(tempstring);
from = Integer.parseInt(st.nextToken());
to = Integer.parseInt(st.nextToken());
sfte.setLocation(new Point(from,to));
return;
}

//Returns true if the argument feature line contains a comment
//The comment is identified by whatever the comment mark is set to
protected boolean containsComment(String line)
{
    if(line.length() > (DESCRIPT_COL) ) //excludes blank comments
    {
        if((line.substring(DESCRIPT_COL-1)).startsWith(COMM_MARK)
        )
        {
            return true;
        }
    }
    return false;
}

//Returns true if the argument feature line contains a qualifier
//i.e. if it has something in the descript column other than a space or a comment mark
protected boolean containsQualifier(String line)
{
    if(line.length() >= (DESCRIPT_COL) ) //includes even one-character
    comments
    {
        if(!((line.substring(DESCRIPT_COL-
1)).startsWith(COMM_MARK) ) && (line.charAt(DESCRIPT_COL-1) != ' '))
        {
            return true;
        }
    }
}

```

```

        return false;
    }

//Returns the qualifier from the argument line
protected String extractQualField(String line)
{
    return chopOffLastChar(line.substring(DESCRIPT_COL-1));
}

//Returns the comment from the argument line
protected String extractCommField(String line)
{
    return line.substring( ( DESCRIPT_COL-1+COMM_MARK.length() ) );
}

//Chops off the last character of a string
protected String chopOffLastChar(String line)
{
    int last_index = line.length() - 1;
    line = line.substring( 0, last_index );
    return line;
}

//-----
// ParseFT method

//Parses a feature table (a vector of strings) and returns a vector of SwissFTEntry
//objects
protected Vector parseFT(Vector ftable) throws Exception
{
    Vector features = new Vector();
    int ftablesize = ftable.size();

    SwissFTEntry sfte = null;
    Vector tempq = null, tempc = null;
    StringTokenizer st;
    String line, temps;

```

```

boolean blankline = false;

for(int count = 0; count<ftablesize; count++)
{
    line = (String) ftable.elementAt(count);
    if(isFeatureLine(line))
    {
        if(isNewFeature(line))
        {
            if(sfte != null)
            {
                compileEntry(sfte, tempq, tempc);
                features.addElement(sfte);
                sfte = null;
                tempq = null;
                tempc = null;
            }

            sfte = new SwissFTEntry();
            extractMinimumFTDetails(sfte, line);
            if(containsQualifier(line))
            {
                tempq = new Vector();

tempq.addElement(extractQualField(line));
            }
            else if(containsComment(line))
            {
                tempc = new Vector();

tempc.addElement(extractCommField(line));
            }
            else if(isFeatureLine(line))
            {
                if(containsQualifier(line))
                {
                    if(tempq == null)
                    {
                        throw new
Exception("Abnormal Feature Format at line " + count);
                    }
                }
            }
        }
    }
}

```

```

tempq.addElement(extractQualField(line));
    }
    else if(containsComment(line))
    {
        if(tempc == null) tempc = new
Vector();

tempc.addElement(extractCommField(line));
    }
    else blankline = true; //debug
    }
} // if isFeatureLine

} //end for
if(sfte != null) //pick up the last one
{
    compileEntry(sfte, tempq, tempc);
    features.addElement(sfte);
    sfte = null;
    tempq = null;
    tempc = null;
}

return features;
} //end function

//=====
//Methods for translateToFlat

//General methods

//Main method. Presents the DBSeqInfo object as a SwissProt entry
protected Vector presentSwissInfo(SwissInfo swissinfo)
{
    if(swissinfo.hasFeatures())
    {
        Vector features = swissinfo.getFeatures();

        trimFeatures(features); //Edits original
    }
}

```

```

        sortFTEntries(features);           //Sorts original
        features = FTtoSwiss(features);     //Returns new array in

    Vector alltext = swissinfo.getOriginalInfo();

    Vector swiss_text = replaceFTLines(alltext, features);

    return swiss_text;
}
else return swissinfo.original_info;
}

protected Vector replaceFTLines(Vector alltext, Vector features)
{
    if( lineExists(FT_MARK, alltext) )
    {
        return replaceLines(FT_MARK, alltext, features);
    }
    else
    {
        int insertpoint = 0;

        if(lineExists(SEQ_MARK, alltext))
        {
            return insertBeforeLine(SEQ_MARK, alltext, features);
        }
        else
        {
            int fsize = features.size();

            for(int i = 0; i<fsize; i++)
            {
                alltext.addElement( features.elementAt(i) );
            }
            return alltext;
        }
    }
}

protected Vector insertBeforeLine(String descriptLine, Vector alltext, Vector insertion)
{
    Vector target = new Vector();

    int insertpoint = findLineIndex(SEQ_MARK, alltext);

    for(int i = 0; i<insertpoint; i++)
    {
        target.addElement( alltext.elementAt(i) );
    }

    int fsize = insertion.size();

    for(int i = 0; i<fsize; i++)
    {
        target.addElement( insertion.elementAt(i) );
    }

    int alltextsize = alltext.size();

    for(int i = insertpoint; i<alltextsize; i++)
    {
        target.addElement( alltext.elementAt(i) );
    }

    return target;
}

//Replaces lines starting with descriptLine in original with those in the replacement
protected Vector replaceLines(String descriptLine, Vector original, Vector replacement)
{
    Vector target = new Vector();

    int origsize = original.size(); int repsize = replacement.size();

    int origstart = findLineIndex(descriptLine, original);

    int origend = findLastLineIndex(descriptLine, original);

    for(int i = 0; i<origstart; i++)
    {
        target.addElement( original.elementAt(i) );
    }
}

```

```

        for(int i = 0; i<resize; i++)
        {
            target.addElement( replacement.elementAt(i) );
        }

        for(int i = (origend + 1); i<origsize; i++)
        {
            target.addElement( original.elementAt(i) );
        }

        return target;
    }

//-----
//Methods specifically to do with constructing the feature table

//Adds space until the specified column is reached

protected String addSpaceUntil(String target, int stopBefore)
{
    int space_needed=stopBefore-target.length()-1;
    for(int i=0; i<space_needed; i++)
    {
        target+=" ";
    }
    return target;
}

//Append a qualifier at the description column

protected String appendQual(String target, String qual)
{
    target = addSpaceUntil(target, DESCRIPT_COL);

    target += (qual.toUpperCase() + ",");

    return target;
}

//Append a last qualifier (i.e. terminated with a '.' rather than a ',')

```

```

protected String appendLastQual(String target, String qual)
{
    target = addSpaceUntil(target, DESCRIPT_COL);

    target += (qual.toUpperCase() + ".");

    return target;
}

//Append a comment at the description column

protected String appendComm(String target, String comm)
{
    target = addSpaceUntil(target, DESCRIPT_COL);

    target +=(COMM_MARK+comm);

    return target;
}

//Construct a minimal line i.e. key/type and location

protected String minimalLine(SwissFTEntry sfte)
{
    String ml = new String(FT_MARK);

    ml=addSpaceUntil(ml,KEY_COL);

    ml+=sfte.getType().toUpperCase();

    String from = null, to = null;

    from=Integer.toString(sfte.getLocation().x);

    ml=addSpaceUntil( ml,( FROM_END_COL-from.length() ) );

    ml+=from;

    to=Integer.toString(sfte.getLocation().y);

    ml=addSpaceUntil( ml,( TO_END_COL-to.length() ) );
}

```

```

        ml+=to;
        return ml;
    }
//Convert an entry to lines of SwissProt format
protected Vector EntryToSwiss(SwissFTEntry sfte)
{
    Vector ft = new Vector();

    String first_line = minimalLine(sfte);

    Vector tempv = null; int vsize;

    String tempstring = null; boolean first_line_not_filled=true;

    if(sfte.qualifiersExist())
    {
        tempv = sfte.getQualifiers();

        vsize = tempv.size();

        if(vsize==1)
            first_line=appendLastQual(first_line, (String)
tempv.elementAt(0) );
        else
            first_line=appendQual(first_line, (String)
tempv.elementAt(0) );

        ft.addElement(first_line);

        first_line_not_filled=false;

        for(int i=1; i<(vsize); i++)
        {
            tempstring = new String("FT");

            if(i==(vsize-1))
                tempstring=appendLastQual(tempstring,
(String) tempv.elementAt(i) );
            else

```

```

tempstring=appendQual(tempstring, (String)
tempv.elementAt(i));

            ft.addElement(tempstring);
        }
    }
    if(sfte.commentsExist())
    {
        tempv = sfte.getComments();

        vsize = tempv.size();

        if(first_line_not_filled)
        {
            first_line=appendComm(first_line, (String)
tempv.elementAt(0) );

            ft.addElement(first_line);

            first_line_not_filled=false;
        }
        else
        {
            tempstring = new String(FT_MARK);

            tempstring = appendComm(tempstring, (String)
tempv.elementAt(0));

            ft.addElement(tempstring);
        }

        for(int i=1; i<(vsize); i++)
        {
            tempstring = new String(FT_MARK);

            tempstring = appendComm(tempstring, (String)
tempv.elementAt(i));

            ft.addElement(tempstring);
        }
    }
    if(first_line_not_filled)
    {

```

```

        ft.addElement(first_line);
    }
    return ft;
}

//Convert a vector of feature table entry objects into a SwissProt feature table
protected Vector FTtoSwiss(Vector features)
{
    Vector swiss_features = new Vector();

    Vector tempv = null; int vsize = 0;

    int features_size = features.size();

    for(int i=0; i<features_size; i++)
    {
        tempv = EntryToSwiss( (SwissFTEntry) features.elementAt(i) );

        vsize = tempv.size();

        for(int j=0; j<vsize; j++)
        {
            swiss_features.addElement(tempv.elementAt(j));
        }
    }
    return swiss_features;
}

//-----
//public class FeatureIntegrityChecker extends SwissProtTranslator
//{

//This function will be shed after debugging

/*
    public FeatureDossier compileDossierOn(SwissFTEntry sfte)
    {
        FeatureIntegrityReport fir = new FeatureIntegrityReport();

        FeatureDossier fd = new FeatureDossier(fir, sfte);

        boolean t = false;

```

```

        t = checkLocation(sfte.getLocation());

        fd.report.location_okay = t;

        fd.report.type_okay = checkType(sfte.getType());

        if(sfte.qualifiersExist())
        {
            fd.report.qualifiers_okay = checkQualifiers(sfte.getQualifiers());
        }
        else
            fd.report.qualifiers_okay = true;

        if(sfte.commentsExist())
        {
            fd.report.comments_okay = checkQualifiers(sfte.getComments());
        }
        else
            fd.report.comments_okay = true;

        fd.sfte = sfte;

        return fd;
    }

*/
//=====
//Methods for trimFeatures
//Methods for integrity checking and trimming of the DBSeqInfo object

//Field checking methods

protected boolean checkLocation(Point loc)
{
    return ( (loc.x >= loc.y) || (loc.x <= MAX_FROM_LENGTH) || (loc.y <=
MAX_TO_LENGTH) );
}

protected boolean checkType(String type)
{
    return (type.length() <= MAX_TYPE_LENGTH);
}

```

```

protected boolean checkQualifiers(Vector quals)
{
    int qualsize = quals.size();
    for(int i=0; i<qualsize; i++)
    {
        if( ( (String) quals.elementAt(i) ).length() >
MAX_QUALIFIER_LENGTH )
            return false;
    }
    return true;
}

protected boolean checkComments(Vector comments)
{
    int commentsize = comments.size();
    for(int i = 0; i<commentsize; i++)
    {
        if( ( (String) comments.elementAt(i) ).length() >
MAX_COMMENT_LENGTH )
            return false;
    }
    return true;
}

//Trims the fields of a SwissFeatureTable entry so that it can be converted
//into SwissProt format. i.e. makes sure each field in small enough to
//fit into the space allocated for it in the flat file format

protected void trimAllFields (SwissFTEntry sfte)// throws Exception
{
    if(! checkLocation(sfte.getLocation() ) )
    {
        sfte.setLocation(new Point(0,0));
    }
    if(! checkType(sfte.getType() ) )
    {
        sfte.setType( trimLine ( sfte.getType(), MAX_TYPE_LENGTH ) );
    }
}

```

```

if( sfte.qualifiersExist() )
{
    if(! checkQualifiers(sfte.getQualifiers() ) )
    {
        sfte.setQualifiers( trimLines ( sfte.getQualifiers(),
MAX_QUALIFIER_LENGTH ) );
    }
}
if( sfte.commentsExist() )
{
    if(! checkComments(sfte.getComments() ) )
    {
        sfte.setComments( trimLines ( sfte.getComments(),
MAX_COMMENT_LENGTH ) );
    }
}

/*
//Remove ...
if( compileDossierOn(sfte).report.toBoolean() == false )
{
    throw new Exception("Integrity checker fault");
}
*/
//... remove.
}

//Trim line to specified length

public String trimLine(String text, int max_length)
{
    if(text.length() > max_length)
    {
        text = text.substring(0, max_length);
    }
    return text;
}

//Trim lines to specified length

public Vector trimLines(Vector textv, int max_length)
{
    int textvsize = textv.size();
    String temps;
}

```

```

        for(int i = 0; i<textvsize; i++)
        {
            temps = (String) textv.elementAt(i);
            if(temps.length() > max_length)
            {
                temps = temps.substring(0, max_length);
                textv.setElementAt(temps, i);
            }
        }
        return textv;
    }
}

```

```

/*
 * CLASS: TestCanvas
 *
 * This class defines an object that acts as a 'dummy' or
 * 'blank' component. The TestCanvas is a simple object that
 * is takes up the required amount of space in a container's
 * layout. TestCanvas, and the identical DummyCanvas, are used
 * as spacers in grid and gridbag layouts.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;

public class TestCanvas extends Canvas
{
    int w,h;

    public TestCanvas(int w, int h)
    {
        this.w = w;
        this.h = h;
    }
}

```

```

public TestCanvas()
{
    this(200,100);
}

public Dimension preferredSize()
{
    return new Dimension(w,h);
}

public Dimension minimumSize()
{
    return preferredSize();
}
}

```

```

/*
 * CLASS: TestPanel
 *
 * This class defines a panel the preferred size of which can
 * be set on construction. It used in the CustomizeDoS class as
 * a container that cannot be reduced in size by a layout
 * manager.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;

public class TestPanel extends Panel
{
    private int w;
    private int h;

    public TestPanel(int w, int h)
    {
        this.w = w;
        this.h = h;
    }

    public Dimension preferredSize()

```

```

    {
        return new Dimension(w,h);
    }

    public Dimension minimumSize()
    {
        return preferredSize();
    }
}

```

```

/*
 * INTERFACE: Triggerable
 *
 * This interface defines the methods that are implemented
 * by a class that is 'triggered' by a single call, go().
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

interface Triggerable
{
    public void go();
}

```

```

/*
 * CLASS: TwoWayHashtable
 *
 * This class encapsulates a hash table that allows key to
 * be looked up by value as well as values to be looked up
 * keys. This class extends dictionary and contains two
 * private complementary Hashtable objects. This class only
 * works effectively where each key and value are unique.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.util.*;

```

```

public class TwoWayHashtable extends Dictionary implements Cloneable
{
    protected Hashtable to, from;

    public TwoWayHashtable(int initialCapacity, float loadFactor)
    {
        to = new Hashtable(initialCapacity, loadFactor);
        from = new Hashtable(initialCapacity, loadFactor);
    }

    public TwoWayHashtable(int initialCapacity)
    {
        to = new Hashtable(initialCapacity);
        from = new Hashtable(initialCapacity);
    }

    public TwoWayHashtable()
    {
        to = new Hashtable();
        from = new Hashtable();
    }

    //Copy constructor for clone() method

    protected TwoWayHashtable(TwoWayHashtable h)
    {
        this.to = (Hashtable) h.to.clone();
        this.from = (Hashtable) h.from.clone();
    }

    //Implementation of dictionary interface

    public synchronized Enumeration elements()
    {
        return to.elements();
    }

    public synchronized Object get(Object key)
    {
        return to.get(key);
    }
}

```

```

public synchronized Object getKeyFor(Object value)
{
    return from.get(value);
}

public boolean isEmpty()
{
    return to.isEmpty();
}

public synchronized Enumeration keys()
{
    return to.keys();
}

public synchronized Object put(Object key, Object value)
{
    from.put(value, key);
    return to.put(key, value);
}

public synchronized Object remove(Object key)
{
    Object value = to.remove(key);
    from.remove( value );
    return value;
}

public synchronized Object removeElement(Object value)
{
    Object key = from.remove( value );
    to.remove( key );
    return key;
}

public int size()
{
    return to.size();
}

//Generic Object features
public synchronized Object clone()
{
    return new TwoWayHashtable(this);
}

public synchronized String toString()
{
    return "to :\n" + to.toString() + "\nfrom :\n" + from.toString();
}

//Other features

public synchronized boolean contains(Object value)
{
    return to.contains(value);
}

public synchronized boolean containsKey(Object key)
{
    return to.containsKey(key);
}
}

/*
 * CLASS: TypeGProperties
 *
 * This class holds information about the graphic properties
 * associated with a particular feature type/key in the
 * DiagramOfFeatures. These properties include the shape,
 * height and colour of the feature, and whether it is
 * designated visible.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

import java.awt.*;

public class TypeGProperties implements Cloneable
{
    public Color fill_color;
    public Color line_color;
    public int shape;
}

```

```

public int height;
public boolean visible;
public static final int LOW = 10;
public static final int MEDIUM = 20;
public static final int HIGH = 30;

public TypeGProperties()
{
    this(Color.red, MyDrawnShape.RECTANGLE, LOW, true);
}

public TypeGProperties(Color fill_color, int shape, int height, boolean visible)
{
    this.fill_color = fill_color;
    this.line_color = Color.black;
    this.shape = shape;
    this.height = height;
    this.visible = visible;
}

public Object clone()
{
    return new TypeGProperties(this.fill_color, this.shape, this.height, this.visible);
}

public String toString()
{
    return new String("Fillcolor: " + fill_color.toString());
}
}

/*
 * CLASS: gblHelper
 *
 * This class contains various forms of the constrain() helper method
 * that makes using the GridBagLayout more convenient. The constrain
 * methods are based on those described in "Java in a Nutshell" by
 * David Flanagan, O'Reilly Associates, First Edition.
 * @author William Valdar
 * @version 1.0 (September 1997)
 */

```

```

import java.awt.*;

public class gblHelper{

    //No constructors

    public static void constrain(Container container, Component component,
                                int gridx,int gridy,int grid_width, int
grid_height){
        constrain(container, component,
                gridx, gridy, grid_width, grid_height,
                GridBagConstraints.NONE, GridBagConstraints.NORTHWEST,
                0.0,0.0,0,0,0,0);
    }

    public static void constrain(Container container, Component component,
                                int gridx,int gridy,int grid_width, int grid_height,
                                int fill, int anchor, double weight_x, double weight_y){
        constrain(container, component,
                gridx, gridy, grid_width, grid_height,
                fill, anchor,weight_x,weight_y,
                0,0,0,0);
    }

    public static void constrain(Container container, Component component,
                                int gridx,int gridy,int grid_width, int grid_height,
                                int top, int left, int bottom, int right){
        constrain(container, component,
                gridx, gridy, grid_width, grid_height,
                GridBagConstraints.NONE, GridBagConstraints.NORTHWEST,
                0.0,0.0,top,left,bottom,right);
    }

    public static void constrain(Container container, Component component,
                                int gridx,int gridy,int grid_width, int grid_height,
                                int border){
        constrain(container, component,
                gridx, gridy, grid_width, grid_height,
                GridBagConstraints.NONE, GridBagConstraints.NORTHWEST,
                0.0,0.0,border,border,border,border);
    }
}

```

```

    }
    public static void constrain(Container container, Component component,
                               int gridx,int gridy,int grid_width, int grid_height,
                               Insets insets){
        Insets i=(Insets)insets.clone();
        constrain(container, component,
                 gridx, gridy, grid_width, grid_height,
                 GridBagConstraints.NONE, GridBagConstraints.NORTHWEST,
                 0.0,0.0,i);
    }

    public static void constrain(Container container, Component component,
                               int gridx,int gridy,int grid_width, int grid_height,
                               int fill, int anchor, double weight_x, double weight_y,
                               int top, int left, int bottom, int right){
        GridBagConstraints c=new GridBagConstraints();
        c.gridx=gridx; c.gridy=gridy;
        c.gridwidth=grid_width; c.gridheight=grid_height;
        c.fill=fill;
        c.anchor=anchor;
        c.weightx=weight_x; c.weighty=weight_y;
        if(top+bottom+left+right>0)
            c.insets=new Insets(top,left,bottom,right);

        ((GridBagLayout)container.getLayout()).setConstraints(component, c);
        container.add(component);
    }

    public static void constrain(Container container, Component component,
                               int gridx,int gridy,int grid_width, int grid_height,
                               int fill, int anchor, double weight_x, double weight_y,
                               Insets insets){
        GridBagConstraints c=new GridBagConstraints();
        c.gridx=gridx; c.gridy=gridy;
        c.gridwidth=grid_width; c.gridheight=grid_height;
        c.fill=fill;
        c.anchor=anchor;
        c.weightx=weight_x; c.weighty=weight_y;
        c.insets=insets;

        ((GridBagLayout)container.getLayout()).setConstraints(component, c);
        container.add(component);
    }

```