DESIGN AND IMPLEMENTATION OF A SPATIAL INDEX STRUCTURE FOR A SPATIAL DATABASE WITH JAVA

By

VEERA ASVASERMCHAROEN

Bachelor of Science

King Mongkut's Institute of Technology Ladkrabang

Bangkok, Thailand

1991

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1996

DESIGN AND IMPLEMENTATION OF A

SPATIAL INDEX STRUCTURE FOR A

SPATIAL DATABASE WITH JAVA

Thesis Approved:

H. Lu

Thesis Adviser

J Chandler

Blayne E. Mayfield

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

It is impossible to thank all those individuals who helped me during my graduate program. The instructors and my friends in the Department of Computer Science at OSU, who gave me encouragement and assistance.

My deepest appreciation goes to Dr. Huizhu Lu, my major advisor. She patiently guided me throughout my thesis, always helping me when needed by giving me exemplary advice. My progress and success in this thesis would have been impossible without her. My sincere appreciation extends to my other committee members Dr. G.E. Hedrick, Dr. Blayne E. Mayfield, and Dr. J.P. Chandler, whose guidance, assistance, and encouragement were also invaluable. I would like to extend my deepest appreciation to Mr. Ross Fenske for his help and advice that helped me impove this paper.

Throughout my life, my parents and my brother have provided constant support, love, and understanding. I would also like to give my special and deepest appreciation to them.

Finally, I would like to thank the Department of Computer Science for supporting me during my past two years of study.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER I

INTRODUCTION

1.1 Background

Information or data is an important part of many businesses, research, and education. A good Database Management System (DBMS) is required for searching, processing, transforming, and storing data. In the past, DBMS did not handle multi-dimensional data such as boxes, polygons, lines, or even points in a multi-dimensional space efficiently.

One of the future requirements for databases is the ability to support multi-dimensional and spatial data for applications, such as, Geographic Information System (GIS), Very Large-Scale Integrated (VLSI) design, and Computer-Aided Design (CAD).

The spatial data structure, the R-tree, which is proposed in [Guttman, 1984], is designed to handle multi-dimensional point and range data. An R-tree is not restricted to storing multi-dimensional points, but can store multi-dimensional spatial objects directly. A spatial data object is represented by its minimum bounding rectangle (MBR) [Theodoridis and Sellis, 1994]. The R-tree data structure is a height-balanced tree that consists of intermediate and leaf nodes. The descriptions of data objects are stored in leaf nodes, while intermediate nodes are built by grouping the rectangles at lower levels. Data objects can overlap each other, cover each other, or be disjointed completely. In an information sharing world, information from only one computer system is not enough. It is vital to be able to share information between different computer systems.

The Internet, an enormous network consisting of millions of hosts from many organizations and countries around the world, is the world's largest network. This network consists of thousands of business, research, government, and educational networks. This Internet provides its users with immediate information and communication facilities. Many services such as electronic mail, remote login, file transfer, discussion forums, database searching, and distributed information systems become an important part in many businesses.

Using information on the Internet, the World Wide Web (WWW, or "Web" for short), wide-area information retrieval initiative aiming to give universal access to a large universe of documents, uses a form of expression called hypertext, text with pointers to other text, that connects related information in a web like structure. Combined with multimedia, the resulting Webs of hypermedia have opened new possibilities for communication, and expanded the ways of connecting different systems. The hypermedia is a superset of hypertext, which not only contains links to other pieces of text, but also to other forms of media, like, sounds, images, and movies.

The Web was introduced to the world in the early 1990's [Gosling and McGilton, 1995]. Recently, the amount of Web data traffic and the number of computers offering information on the Web have increased. However, the contents of the pages (Web pages) lack expressive and interactive qualities. Web content has been devoid of the degree of interactivity offered by many multimedia and hypermedia systems that run on non-networked computers. For example, on non-networked systems we can draw a picture, create an animation with sound, play games, etc. Even though the Web fosters world-wide interconnections between people and information, it does not give people the ability to interact with the information within the same program (i.e., a browser which is a program that allows users to use the Internet's World Wide Web). In 1990, James Gosling started the design of a new programming language without the problems of traditional languages e.g. C, C++. The result is the Java language.

Java is a new object-oriented programming language developed by Sun Microsystems. Java makes it possible for developers to create content that can be delivered to and run by users on their own computers. This language can provide most of the requirements for the programmers. With the delivery platform as the Web page, this software can support a variety of information tasks with true interactivity [Hoff, Shaio, and Starbuck, 1996].

1.2 Purpose of the Study

The major objective of this thesis is to implement a multi-dimensional spatial index data structure, the R-tree, on the Internet by using the Java language. In addition, the author will apply the index structure to access a simple and small spatial database.

This thesis contains five chapters. The second chapter describes the R-tree data structure, and reviews the Java language. The implementation platform and environment are presented in Chapter III. The design and implementation are analyzed in Chapter IV. Finally, conclusions and future work are included in Chapter V.

CHAPTER II

LITERATURE REVIEW

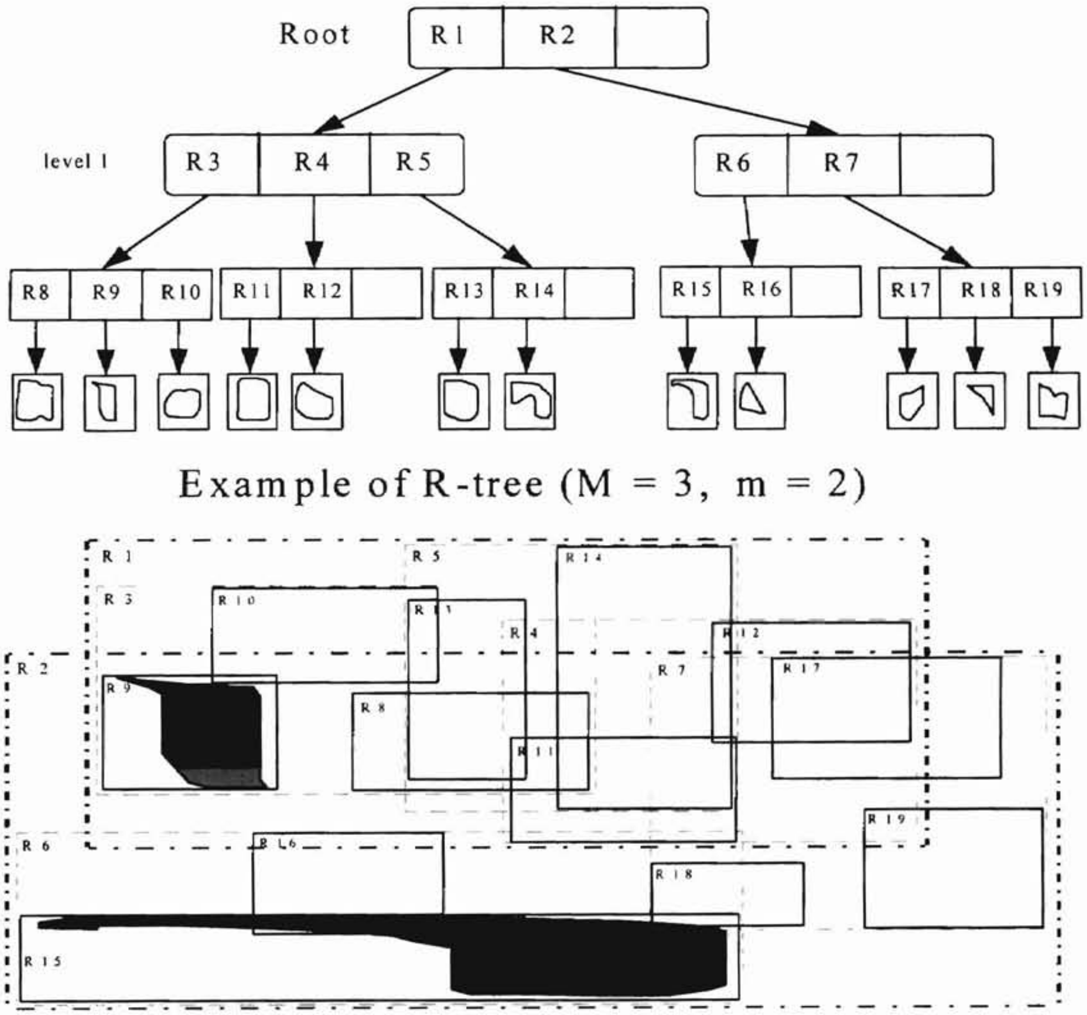
Data structures that can handle multi-dimensional data objects are required in database applications such as Geographic Information System (GIS), Very Large-Scale Integrated (VLSI) design, and Computer-Aided Design (CAD). Traditional data structures, for example B-tree, that support one dimensional data object are not suitable for multi-dimensional data objects. In a multi-dimensional data space, in the simplest form, a point, has at least two coordinates (x, y), and in complex form, a polygon, requires more than two coordinates to represent it.

2.1 R-tree data structure

An R-tree is a high-balanced tree that consists of 2 types of nodes: intermediate and leaf nodes. Pointers in a leaf node point to data objects. Intermediate nodes keep the address of a lower node in the R-tree.

Non-leaf nodes (intermediate nodes) contain entries of the form (R, ptr) where ptr is a pointer to a child node in the R-tree and R is an n-dimensional rectangle that covers all rectangles in the child node. Leaf nodes contain entries of the form (R, obj-id) where obj-id is a pointer to the object description and R is the minimum bounding rectangle of the object. Let 'M' represent the maximum number of entries that can be contained in a node, and 'm' represent the minimum number of entries in a node ($2 \le m \le {}^{M}/_{2}$) then R-tree has these properties [Guttman, 1984] (see Figure 2.1.1).

1. Every leaf node contains between m and M index record unless it is the root.
2. Every non-leaf node has between m and M children unless it is the root.
3. The root node has at least two children unless it is a leaf.
4. All leaves appear on the same level.



Example of R-tree (M = 3, m = 2)

Figure 2.1.1: *The sample of R-tree data structure*

From the Figure 2.1.1 R1, R2, ..., and R7 are non-leaf nodes, and R8, R9, ..., and R19 are leaf nodes. An R1 contains the MBR of all the nodes R3, R4, R5, R8, R9, R10, R11, R12, R13, and R14, and a pointer to a child node. An R8 contains MBR of an object, and a pointer to the data object.

2.2 Review the Java language

Java is just one part of the integrated set of systems that support World Wide Web (WWW) communication. Java is an entirely separate programming language from the markup scheme for defining hypertext, the Hypertext Markup Language (HTML). Java does not replace HTML; it does not change any work that has been developed on the Web so far.

Java integrates with HTML and the Web through a special HTML element called *APP*, which brings the possibility for Web pages to have Java programs, called applets, on them. An applet is the program that writes in the Java language. These applets are essential software programs that the user's browser automatically downloads (as part of Web page observation) and executes. With graphical input and output possible through the applet displayed on the page, Java extends the Web into higher levels of interactivity.

Java is a high-level programming language, similar to C, C++, Pascal, and Modula-3, but Java is much simpler. All the unnecessary features of high-level programming languages are left out. For example, Java has no pointers, no overloading, no header files, no pre-processor, no pointer arithmetic, no unions, no templates, and no implicit type conversion, etc. Since Java designers were not burdened with compatibility with existing languages, they were able to learn from the experience and mistakes of previous object-oriented languages. They added a few features that C++ does not have, for example, garbage collection and multithreading, and took out the unnecessary features. Garbage collection is the new design to free unused memory, and multithreading is the idea that allows the users to run the concurrent task.

Most things in Java are objects, except for simple types such as numbers and boolean. Java code is organized into classes. Each class defines a set of methods that form the behavior of those objects. A class can inherit behaviors from another class. At the root of the class hierarchy is always class *Objects,* which is different from objects (see Figure 2.2.1). Java supports a single-inheritance class hierarchy. This means that each class can only inherit from one other class at a time. Some languages also allow multiple inheritance, but this may confuse and make the language unnecessarily complicated.

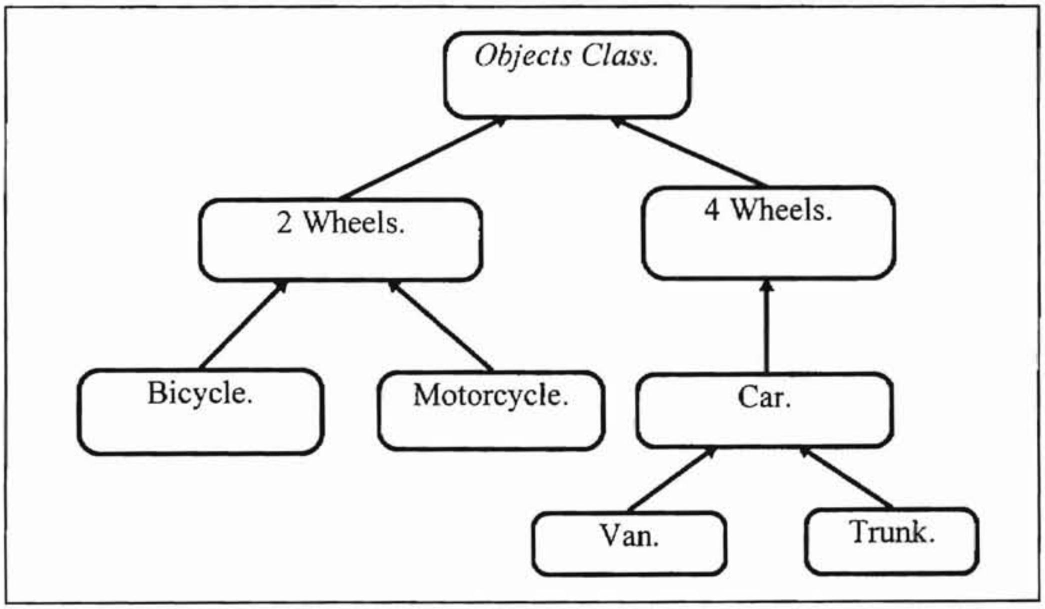


Figure 2.2.1: *The sample hierarchy of Objects class*

When running a Java program, it is first compiled into *byte-codes*. Byte-codes are very similar to machine instructions, so Java programs can be very efficient. Byte-codes are not specific to a particular machine, so Java programs can be executed on many different computers without having to recompile the source codes. Java source programs are compiled to *class files*, which contain the byte-code representation of the program. In a Java class file, all references to methods and instance variables are made by name and are resolved when the code is first executed. A Java interpreter then executes the byte-code.

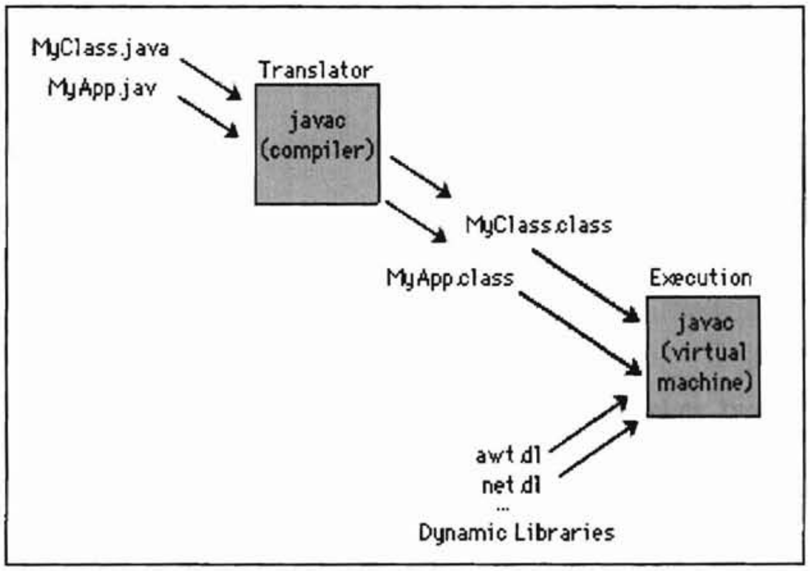


Figure 2.2.2: *The Java development environment.* [Source: Aitken, 1996]

There are several advantages of the Java language. The class files do not include the library classes. The Java interpreter will add the library classes when it runs the programs, so the sizes of the class files are very small (see Figure 2.2.2). The Java language keeps classes in separate files. This will allow the users to create tools for other programs easily; for example, we have one file in Java language, *test.java*, that contains

```
class   XClass {
    int      aa;
    float    bb;
         .
         .
         .
};
class   YClass {
    Object      cc;
    Float       dd;
         .
         .
};
```

After compiling *test.java*, the Java compiler makes two new files, XClass.class, and Yclass.class. Although the users compile the Java source codes under different operating system, the users will get the same byte code (see Figure 2.2.3).

The disadvantage of the Java language is that it is slow when compared with C and C++, because the Java compiler cannot compile source code to machine code. The users must use the Java interpreter to interpret the byte-code every time when they want to run the programs (see Figure 2.2.3).

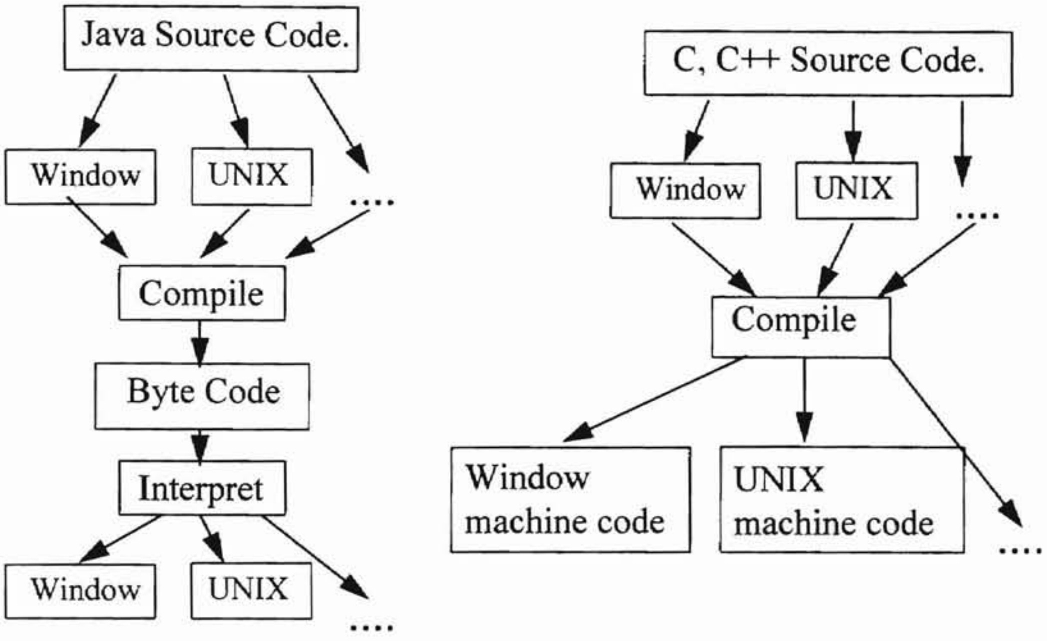


Figure 2.2.3: *The Java language compares with C, and C++.*

2.2.1 Java lang class

The java.lang package contains classes that are fundamental to the design with Java language. The most important class is the Object, which is the root of the class hierarchy (see fig 2.2.1.1). It is necessary to represent a value of primitive type as if it were an object. The wrapper classes Boolean, Character, Integer, Long, Float, and Double serve this purpose. For example an Integer class contains a field whose type is an integer [Gosling and McGilton, 1995]. These classes also provide a number of methods for converting among primitive values, as well as supporting such standard methods. We can use member function "equals" to compare two objects.

The Math class provides commonly used mathematical functions such as sine, cosine, and square root. The classes String and StringBuffer similarly provide commonly used operations on character strings. ClassLoader, Process, Runtime, SecurityManager, and System classes provide "system operations" that manage the dynamic loading of classes, creation of external processes, host environment inquires such as the time of day and enforcement of security policies.

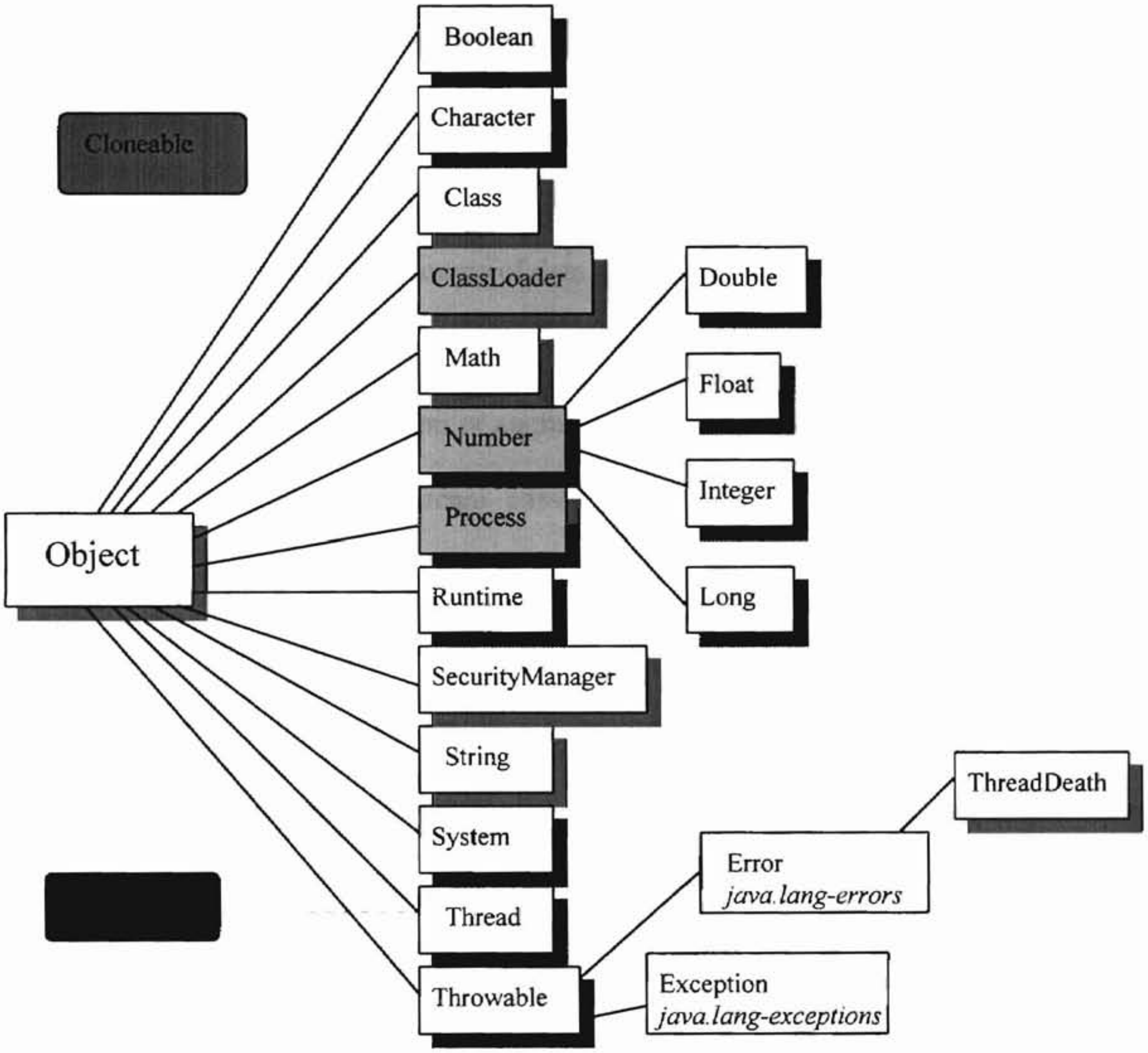


Figure 2.2.1.1: *java.lang package.* [Source: Perkins, 1995]

2.2.2 Java I/O class.

Input and output in Java are organized around the concept of streams. A stream is a sequence of items, usually in 8-bit bytes for reading or writing.

In the java.io package, all input is done through subclasses of the abstract class *InputStream*. All output is done through subclasses of the abstract class *OutputStream*. The *RandomAccessFile* class handles files that allow random access and perhaps intermixed reading and writing of the file (see Figure 2.2.2.1).

For an input stream, the source of data may be a file, a String, an array of bytes, or bytes written to an output stream. There are also "filter input streams" that take data from another input stream and transform or augment the data before delivering it as input. For example, a LineNumberInputStream passes bytes through verbatim but counts line terminators as they are read.

For an output stream, the source of data may be a file, an array of bytes, or a buffer to be read as an input stream. There are also "filter output streams" that transform or augment data before writing it to another output stream.

A File class represents a path name that might identify a particular file within a file system. Certain operations on the file system such as renaming and deleting files are done by this class. A FileDescriptor class represents an abstract indication of a particular file within a file system; such file descriptors are created internally by the Java I/O system.

There are two interfaces, DataInput and DataOutput, that support the transfer of data other than bytes or characters, such as long, integers, floating-point numbers, and strings. The class DataInputStream implements DataInput; the class DataOutputStream implements DataOutput; and RandomAccessFile implements both DataInput and DataOutput.

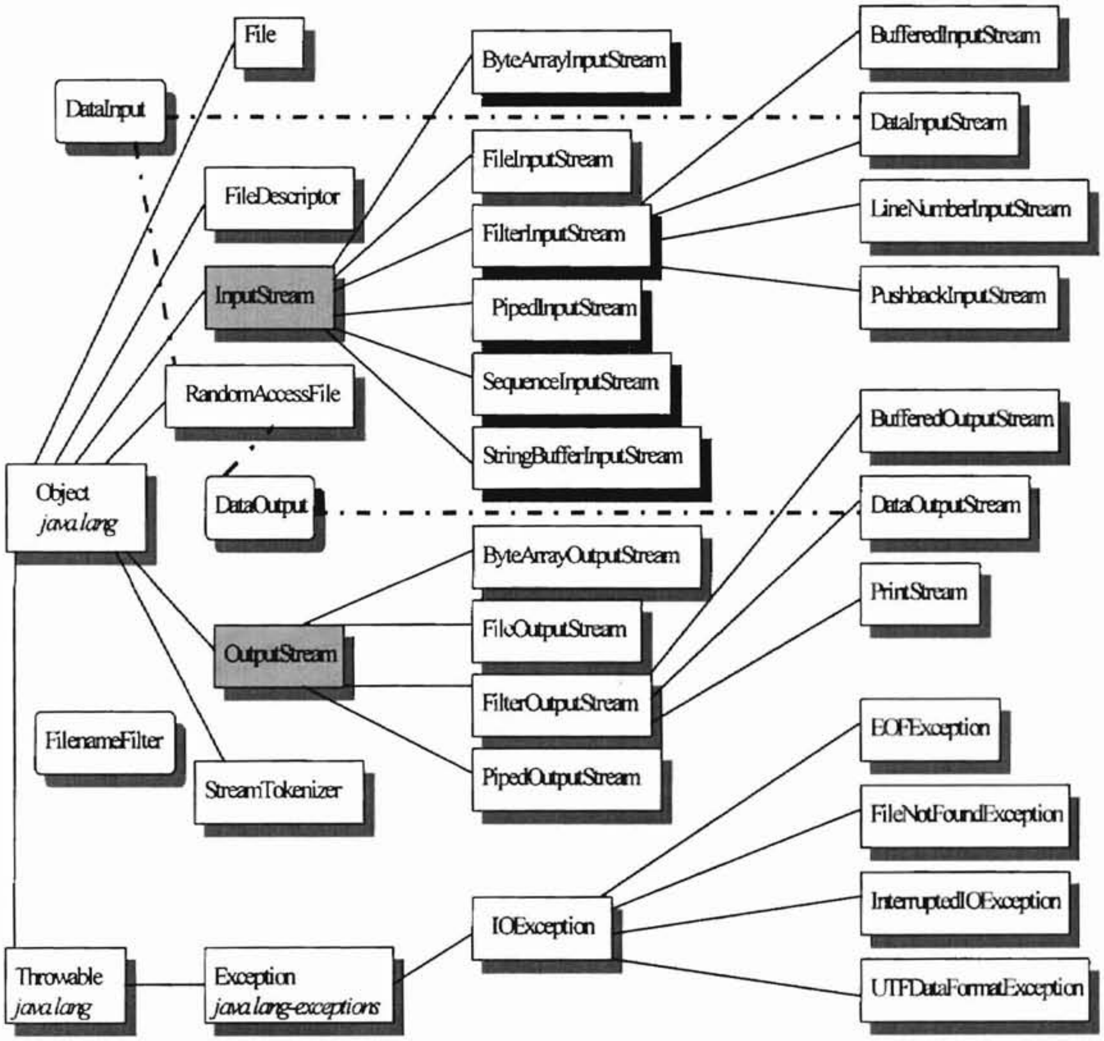


Figure 2.2.2.1: *java.io package.* [Source: Perkins, 1995]

2.2.3 Java awt class.

The java.awt package provides the standard graphical user interface (GUI) elements such as checkboxes, scrollbars, buttons, text fields, and text areas. It also includes containers such as windows, menus, and menu bars, and higher-level components such as dialog boxes. (AWT = Abstract Window Toolkit) (see Figure 2.2.3.1)

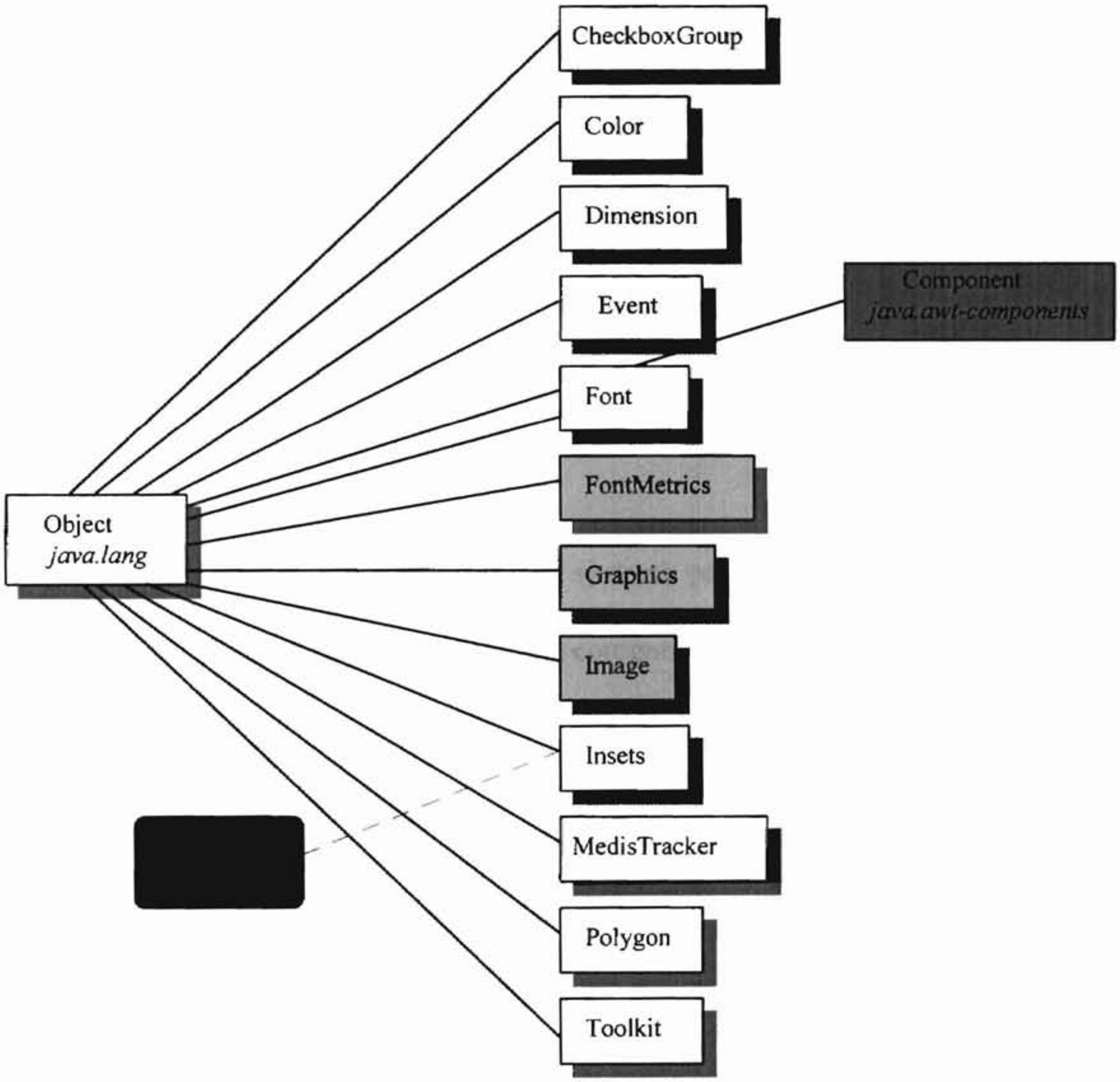


Figure 2.2.3.1: *java.awt package.* [Source: Perkins, 1995]

CHAPTER III

IMPLEMENTATION PLATFORM AND ENVIRONMENT

This section presents a description of the operating system and environment that use for implementation the multi-dimensional database system. The environment of this program is the World Wild Web, so we must choose the operating system that can run the Java language via the Internet. The Solaris operating system is one of the operating systems that has a Java compiler.

3.1 Solaris 2.4

The Solaris is a version of UNIX systems provided by Sun Microsystems for SPARC (SPARC International is one of the computer trademark), and x86 (computer that uses Intel processors). The Solaris 2.x is a full 32-bit operating system based on UNIX System V Release 4 (SVR4). It has extensive functionality in areas such as symmetric multiprocessing (MP) with multithreads (MT), real-time functionality, increased security, and improved system administration. The Solaris 2.x can run on SPARC as well Intel 386, 486, Pentium and other DOS-compatible CPUs, and it has a new Network Information Service, called NIS+. Network Information Service Plus (NIS+) is an upward-compatible version of the NIS name service with simpler hierarchical administration, improved security, and faster updates [Daniel, 1994].

This section shows the main difference between SVR4 and the Solaris operating environment. It points out the features that the Solaris operating environment includes that are not available in SVR4 and a few SVR4 features that are not available in the Solaris operating environment.

Features for the users, the Solaris operating environment incorporates a suite of powerful DeskSet applications to enhance personal productivity. The DeskSet is the software tools that contains Mailtool, xspread (X Spreadsheet), FAX, Addressmanager, Epoch Backup (Backup Software), Image Tools (Image Magick, XV, ...), Word Processing (Interleaf, LaTeX, Editors, ...). All DeskSet applications rely on the drag and drop metaphor, enabling users to carry out complex UNIX commands with a mouse.

As features for the system administrators, the Solaris operating environment offers a variety of new tools to simplify the administration of a distributed computing environment.

As features for application developers, the Solaris operating environment includes a variety of toolkits and features to simplify the development of complex applications with graphical user interfaces.

In a few instances, features in SVR4 were not included in the Solaris operating environment. These features are specific to AT&T® hardware, or features included primarily for backward compatibility with UNIX System V Release 3 (SVR3) features.

The Solaris is one of the operating system that can run the Java language via the Internet World Wild Web.

3.2 World Wide Web (WWW)

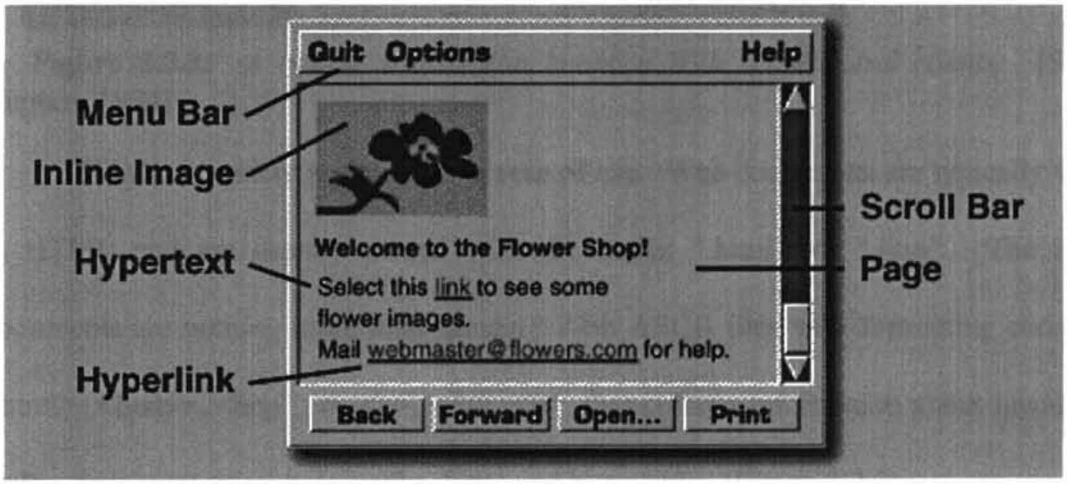
The Internet is the world's largest computer network. It is an international collection of smaller networks and computers. Users on the Internet can exchange electronic mail with each other, copy files from computer to computer (even if those computers are thousands of miles away), play games, access databases of information, etc.

In its short history, the Internet has seen more than its share of revolutions. First came the revolution in global communications with e-mail. After that, File Transfer Protocol (FTP) and Gopher revolutionized information sharing. In the past few years, the World Wide Web has revolutionized the presentation of information. Governments, magazines, television networks, businesses, and other organizations are turning to the Web as a new means of reaching their audience.

The World Wide Web is mostly used on the Internet, but they do not mean the same thing. The Web refers to a body of information, an abstract space of knowledge, while the Internet refers to the physical side of the global network. "Nobody owns the Internet" [Hughes, 1994], although there are companies that help manage different parts of the networks that tie everything together, there is no single governing body that controls what happens on the Internet. The networks within different countries are funded and managed locally according to local policies.

The World Wide Web uses the Internet to transmit hypermedia documents between computer users internationally. It is possible to use the World Wide Web software without having to use the Internet. But Internet access is necessary in order to make full use of and participate in the World Wide Web.

There is no standard way to view the World Wide Web. However, many software interfaces to the Web have similar functions and generally work in the same way no matter what computer or type of display device is used. In fact, many users browser around the Web using text-only interfaces and are able to see all of the textual information with a graphic display device. Although there are many different ways to represent a document on the screen, it is often called a page. Figure 3.2.1 is one of the example of different pages.



Welcome to the Flower Shop!
Select this link to see some flower images.
Mail webmaster@flowers.com for help.

Figure 3.2.1: *A typical Web browser for a graphic user interface.* [Source: Hughes, 1994]

Web software is designed based on a distributed client-server architecture. A Web client is a program which can send requests for documents to any Web server. A Web server is a program that, upon receipt of a request, sends the document requested back to the requesting client. Using a distributed architecture means that a client program may be running on a completely separate machine from that of the server, possibly in another room or even in another country (see Figure 3.2.2). The language that Web clients and servers use to communicate with each other is called the Hypertext Markup Language (HTML).

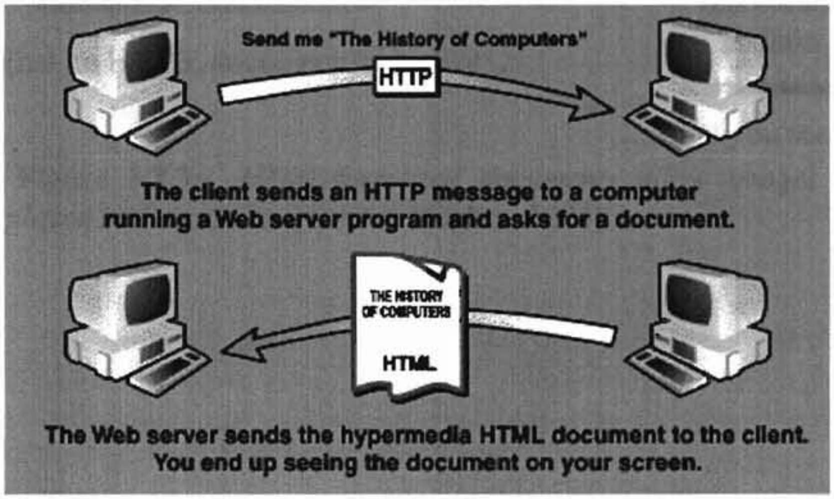
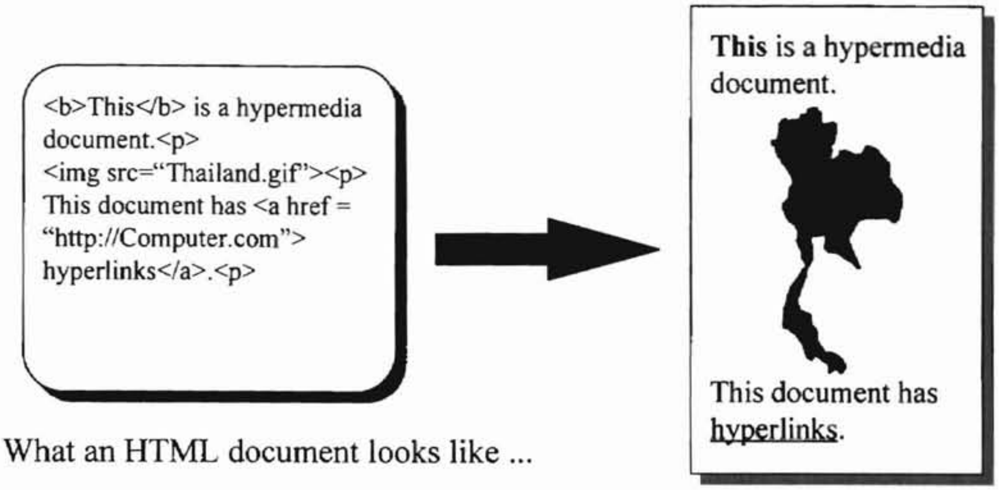

Send me "The History of Computers"
HTTP
The client sends an HTTP message to a computer running a Web server program and asks for a document.
THE HISTORY OF COMPUTERS
HTML
The Web server sends the hypermedia HTML document to the client. You end up seeing the document on your screen.

Figure 3.2.2: *A typical transaction between Web servers and clients.* [Source: Hughes, 1994]

HTML is widely praised for its ease of use. Web documents are typically written in HTML and are usually named with the suffix ".html", or ".htm". The HTML documents are nothing more than standard 7-bit ASCII files with formatting codes, e.g. <html>, <body>, <app ....>, <b>, </b>, <p>, that contain information about layout (text styles, document titles, paragraphs, lists) and hyperlinks. (Figure 3.2.3)



```
<b>This</b> is a hypermedia
document.<p>
<img src="Thailand.gif"><p>
This document has <a href =
"http://Computer.com">
hyperlinks</a>.<p>
```

This is a hypermedia document.

This document has hyperlinks.

What an HTML document looks like ...

... what you see on the screen

Figure 3.2.3: *HTML-formatted documents allow images and hyperlinks to be displayed in documents.* [Source: Hughes, 1994]

CHAPTER IV

DESIGN AND IMPLEMENTATION

4.1 Design a multi-dimensional spatial database system

The main focus of this thesis is to implement a multi-dimensional spatial database application that can run on any platform via the Internet without recompiling the program, so we choose the Java language to implement this application. Java is the new object-oriented (with single inheritance) programming language. It has the property of being able to run an application via the Internet. The application uses the object-oriented approach to contain classes. These classes include: mainClass, StatusBar, ToolBar, HelpDialog, ReportDialog, AboutDialog, NewDialog, ReadURLDialog, ErrorDialog, DisplayDialog, DisplaySearchDialog, ReadFileURL, ConstClass, BranchClass, RootClass, RectClass, NodeClass, SplitClass, IndexClass, and DisplayClass.

The application can be divided into two parts.

1. Implementation of the multi-dimensional spatial data structure.
2. Port and run the application on the Internet by extending the user interface.

4.1.1 The multi-dimensional index structure for spatial databases

The spatial data structure R-tree [Guttman, 1994] is selected to be the spatial index mechanisms to handle multi-dimensional spatial objects in a spatial database. The operations that can be performed on the multi-dimensional spatial database are search, insert, delete, and display. The menu interfaces are implemented to aid novice users in performing various operations easily.

4.1.2 Part of the user interface

We add "java.awt.*" (java Abstract Window Toolkit), "java.io.*" (java Input/Output), "java.lang.*" (java Language), and "java.net.*" (java Networking) to the application. The "java.lang" is the package that contains essential Java classes, including numeric, strings, objects, compiler, and threads. This package is the only package that is imported automatically into every Java program. In this thesis we use this package for implementation string, and mathematics function. The "java.awt" is the package that provides the standard graphical user interface (GUI) elements such as buttons, lists, menus, and text areas. It also includes containers (such as windows and menu bars), and components (such as dialogs for opening and saving files). In this application we use this package to create menu, menu bars, display output, input data, and all user interface. The "java.io" is the package that provides a set of input and output streams used to read and write data to files or other input and output sources at the local site. We cannot use function in this package to read and write files via the Internet. The "java.net" is the package that contains networking classes and a URL connection, TCP sockets, UDP sockets, IP addresses, and a class that represent an Internet address. This package can be used in combination with the classes in "java.io" to read information from the network.

4.2 Implementation

4.2.1 Program Environment

All programs are written in Java programming language. The system that is used to implement the program is a SPARC machine with 64 Mb. The operating system used on this machine is Solaris 2.4, which is a version of UNIX. The Java compiler on Sun Microsystems is used to compile and execute the program. In this program, page size is 1024 bytes. The size of maximum number of entries depends on the dimension and the size of minimum number of entries is equal to $\lceil$maximum number / 2$\rceil$.

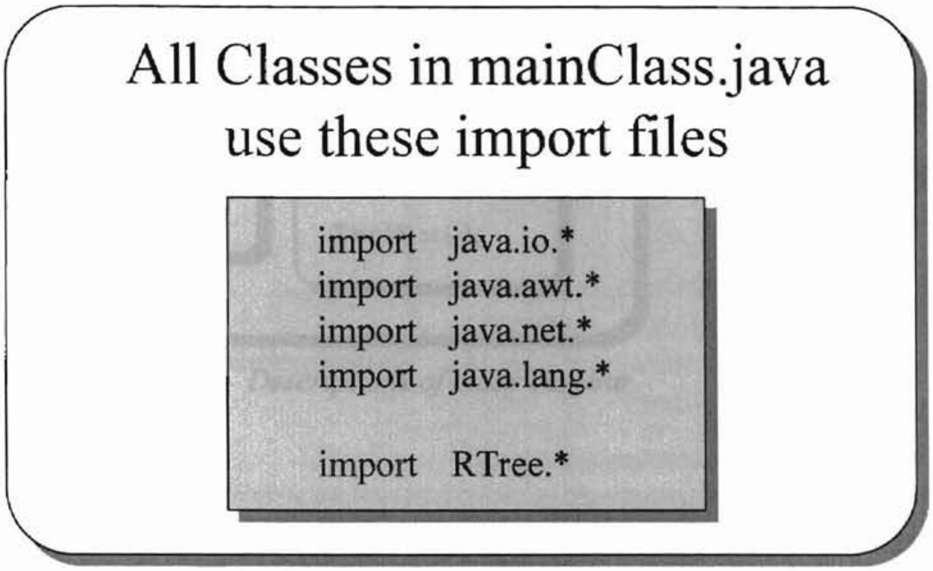
4.3 Description of all classes

The useful feature in object-oriented languages is allowing classes to be related to each other through inheritance. Thus a class can be declared to be a subclass of another class. The subclass does all the things its parent class does, plus some additional features unique to the subclass. The Java language does not have multiple inheritance; that means in the Java language we can inherit class from only one super class. The "extends" means the extended class is the super class. For example, "class mainClass extends Frame" means mainClass is the inheritance class from class Frame, or the class Frame is the super class of the class mainClass. "import" is the reserve word. It means the same as "include" in C, or C++. The following classes have been created for an index structure of the multi-dimensional spatial database.

4.3.1 All classes in mainClass.java

1. class mainClass	extends Frame
2. class StatusBar	extends Panel
3. class ToolBar	extends Panel
4. class HelpDialog	extends Dialog
5. class ReportDialog	extends Dialog
6. class AboutDialog	extends Dialog
7. class NewDialog	extends Dialog
8. class ReadURLDialog	extends Dialog
9. class ErrorDialog	extends Dialog
10. class DisplayDialog	extends Dialog
11. class DisplaySearchDialog	extends Dialog
12. class ReadFileURL	

The following figures are the description of all classes.

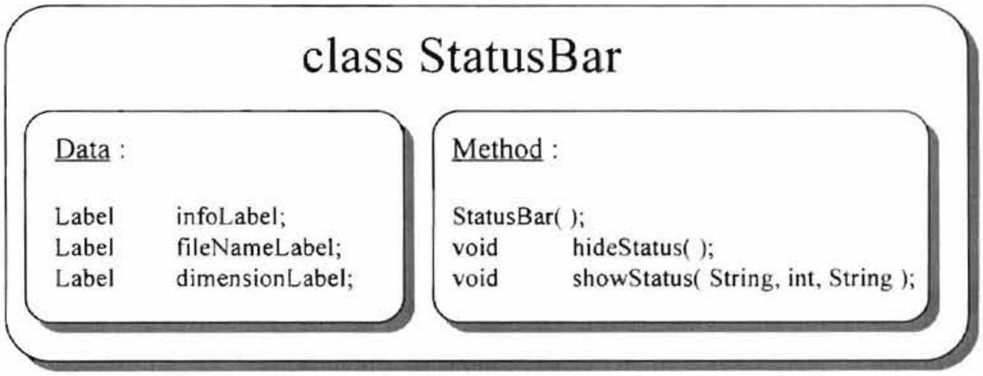


All Classes in mainClass.java use these import files

```
import  java.io.*
import  java.awt.*
import  java.net.*
import  java.lang.*

import  RTree.*
```

Figure 4.3.1.1: *import files for all classes in mainClass.java.*

"import java.io.RandomAccessFile" means the program includes class RandomAccessFile in the package java.io when run this program. "import java.io.*" means the program includes all of the classes in the package java.io.



class StatusBar

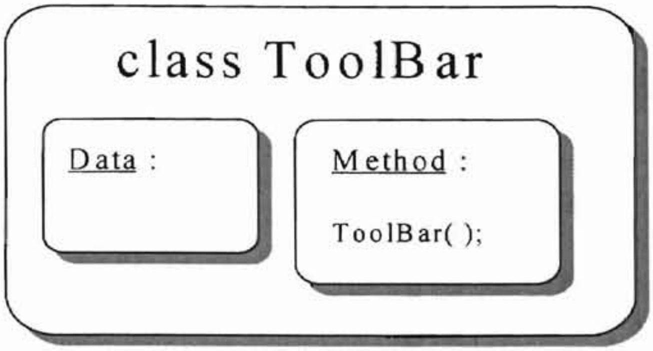
Data :

```
Label     infoLabel;
Label     fileNameLabel;
Label     dimensionLabel;
```

Method :

```
StatusBar( );
void      hideStatus( );
void      showStatus( String, int, String );
```

Figure 4.3.1.2: *Descriptions of class StatusBar.*



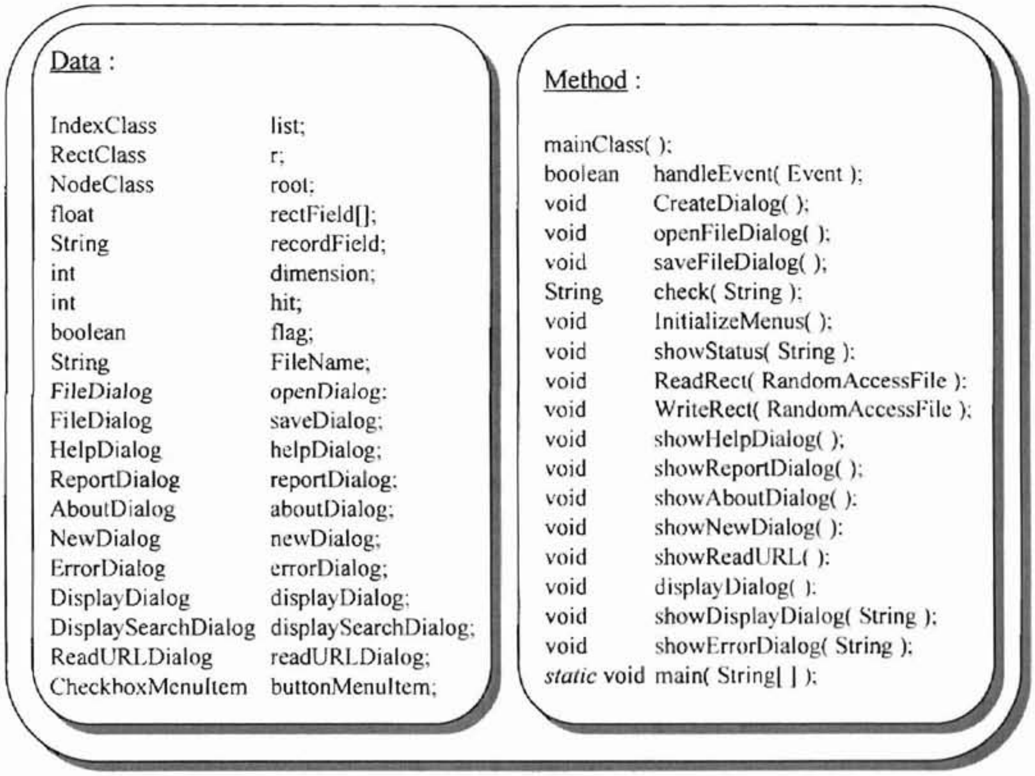
class ToolBar

Data :

Method :

ToolBar( );

Figure 4.3.1.3: *Descriptions of class ToolBar.*



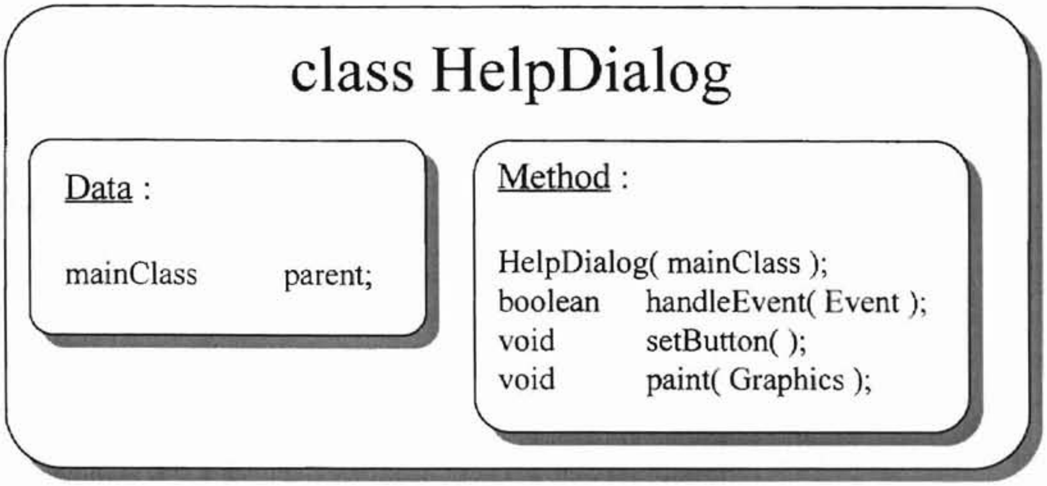
Data :

IndexClass	list;
RectClass	r;
NodeClass	root;
float	rectField[];
String	recordField;
int	dimension;
int	hit;
boolean	flag;
String	FileName;
FileDialog	openDialog;
FileDialog	saveDialog;
HelpDialog	helpDialog;
ReportDialog	reportDialog;
AboutDialog	aboutDialog;
NewDialog	newDialog;
ErrorDialog	errorDialog;
DisplayDialog	displayDialog;
DisplaySearchDialog	displaySearchDialog;
ReadURLDialog	readURLDialog;
CheckboxMenuItem	buttonMenuItem;

Method :

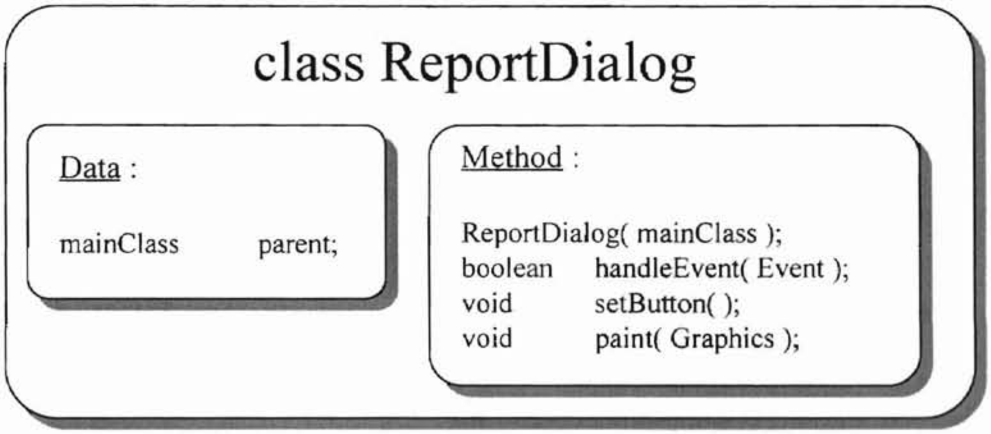
mainClass( );	
boolean	handleEvent(Event);
void	CreateDialog( );
void	openFileDialog( );
void	saveFileDialog( );
String	check(String);
void	InitializeMenus( );
void	showStatus(String);
void	ReadRect(RandomAccessFile);
void	WriteRect(RandomAccessFile);
void	showHelpDialog( );
void	showReportDialog( );
void	showAboutDialog( );
void	showNewDialog( );
void	showReadURL( );
void	displayDialog( );
void	showDisplayDialog(String);
void	showErrorDialog(String);
static void	main(String[ ]);

Figure 4.3.1.4 *Descriptions of class mainClass.*



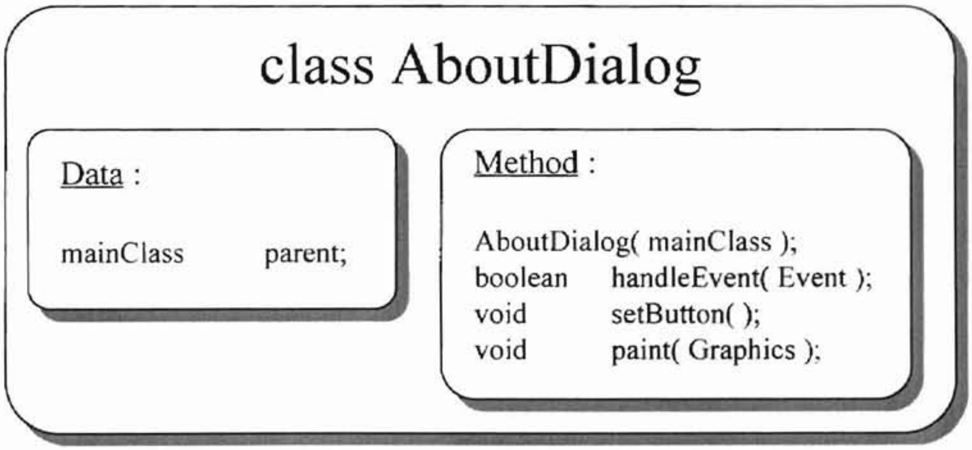
```
class HelpDialog

Data :

mainClass       parent;

Method :

HelpDialog( mainClass );
boolean    handleEvent( Event );
void       setButton( );
void       paint( Graphics );
```

Figure 4.3.1.5: *Descriptions of class HelpDialog.*



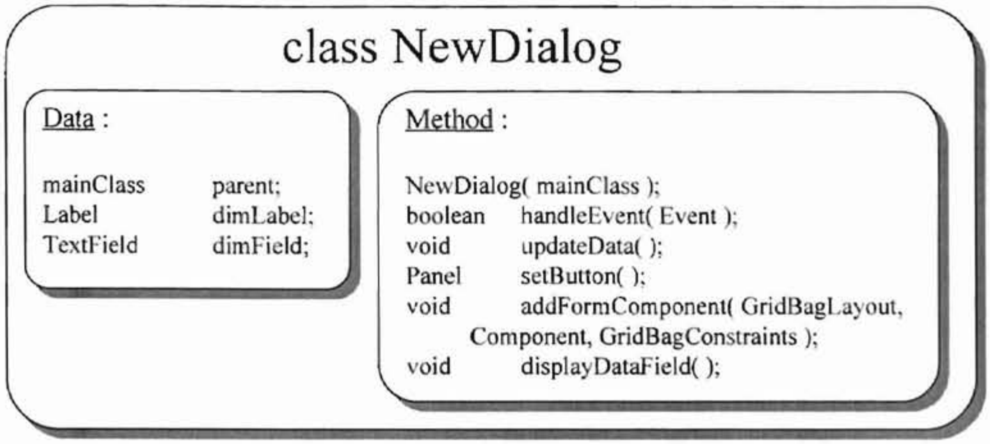
```
class ReportDialog

Data :

mainClass       parent;

Method :

ReportDialog( mainClass );
boolean    handleEvent( Event );
void       setButton( );
void       paint( Graphics );
```

Figure 4.3.1.6: *Descriptions of class ReportDialog.*



```
class AboutDialog

Data :

mainClass       parent;

Method :

AboutDialog( mainClass );
boolean    handleEvent( Event );
void       setButton( );
void       paint( Graphics );
```

Figure 4.3.1.7: *Descriptions of class AboutDialog.*

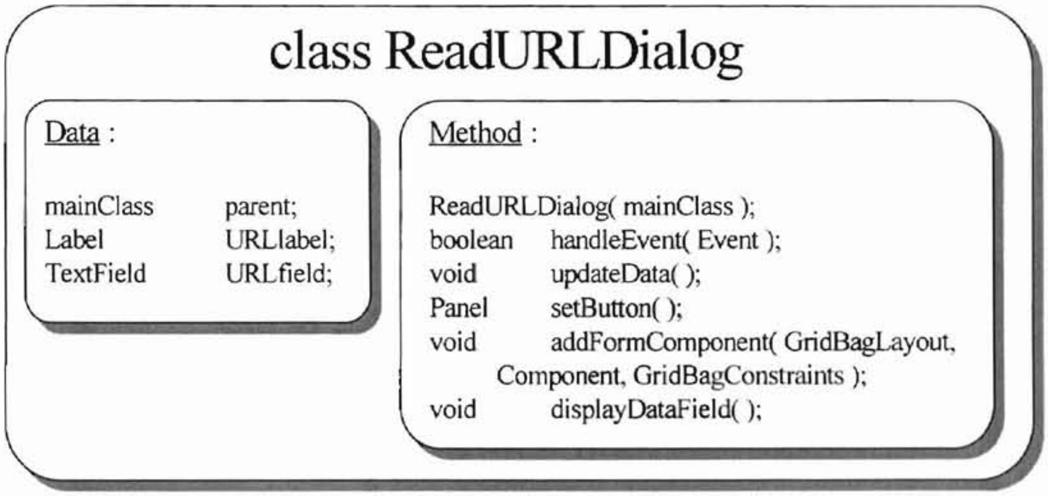


class NewDialog

```
Data :

mainClass      parent;
Label          dimLabel;
TextField      dimField;
```

```
Method :

NewDialog( mainClass );
boolean    handleEvent( Event );
void       updateData( );
Panel      setButton( );
void       addFormComponent( GridBagLayout,
      Component, GridBagConstraints );
void       displayDataField( );
```

Figure 4.3.1.8: *Descriptions of class NewDialog.*

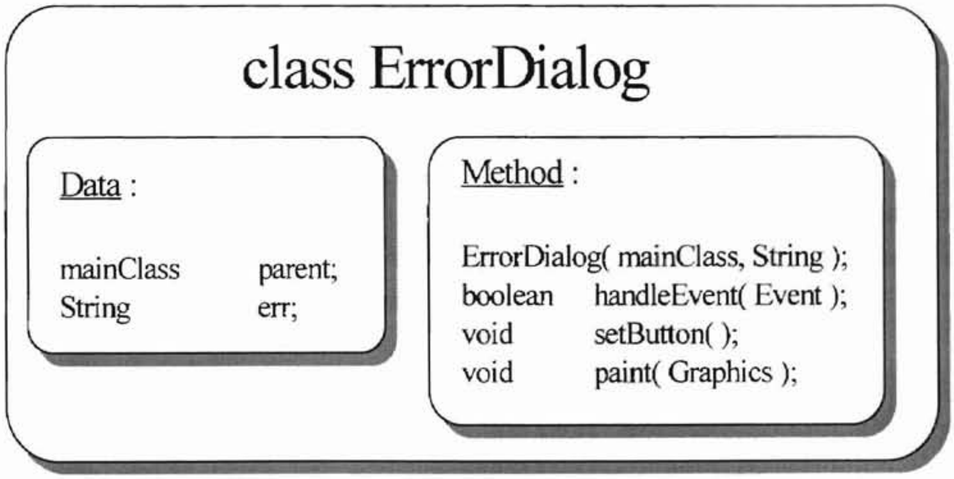


class ReadURLDialog

```
Data :

mainClass      parent;
Label          URLlabel;
TextField      URLfield;
```

```
Method :

ReadURLDialog( mainClass );
boolean    handleEvent( Event );
void       updateData( );
Panel      setButton( );
void       addFormComponent( GridBagLayout,
      Component, GridBagConstraints );
void       displayDataField( );
```

Figure 4.3.1.9: *Descriptions of class ReadURLDialog.*

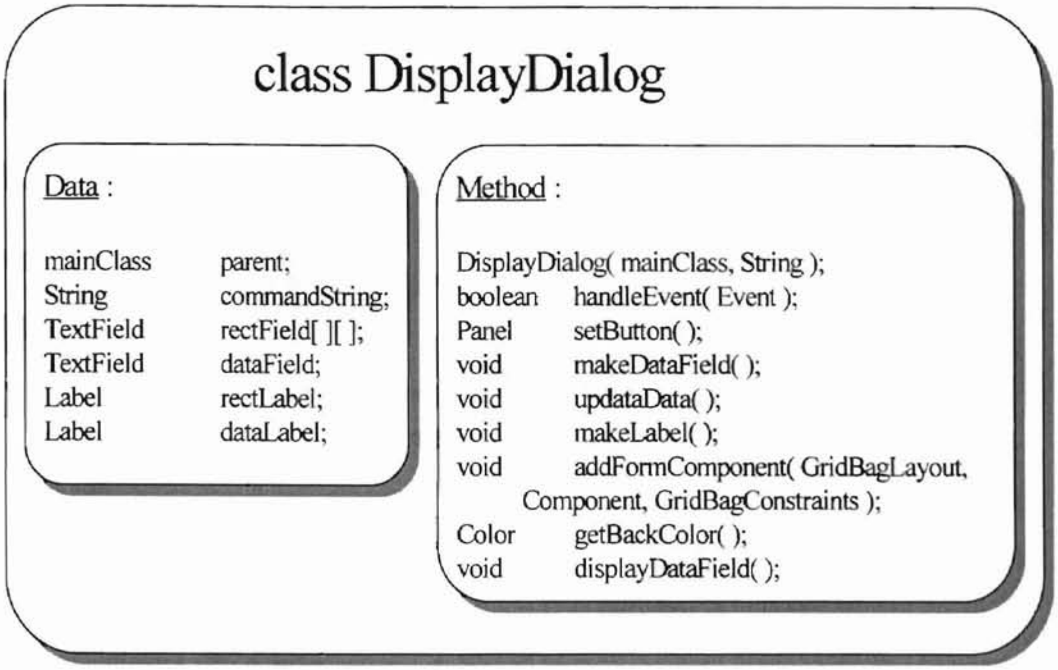


class ErrorDialog

```
Data :

mainClass      parent;
String         err;
```

```
Method :

ErrorDialog( mainClass, String );
boolean    handleEvent( Event );
void       setButton( );
void       paint( Graphics );
```

Figure 4.3.1.10: *Descriptions of class ErrorDialog.*



class DisplayDialog

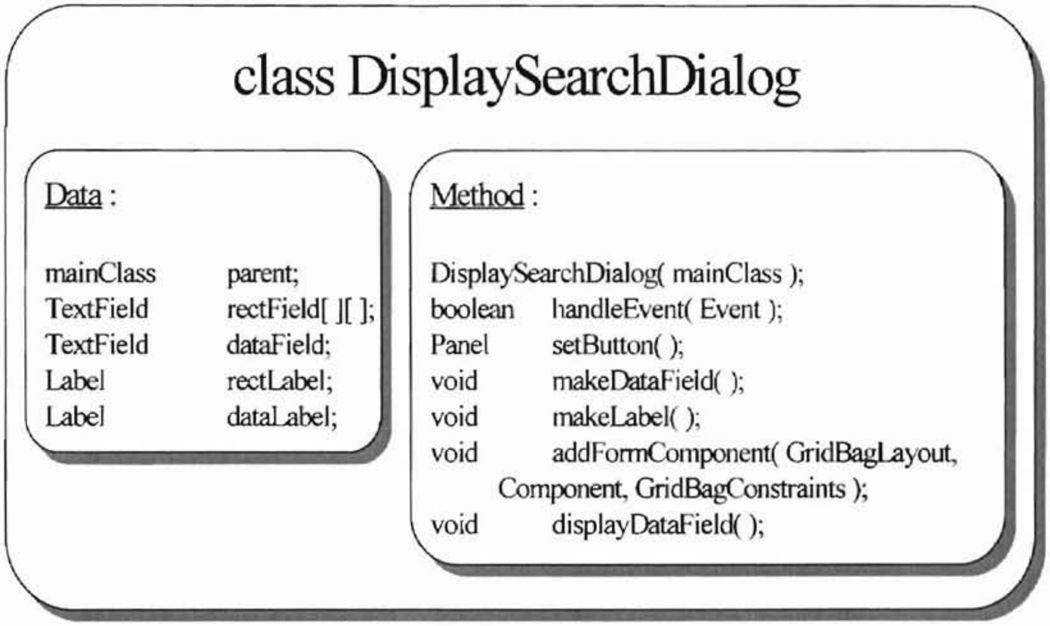
Data :

mainClass	parent;
String	commandString;
TextField	rectField[ ][ ];
TextField	dataField;
Label	rectLabel;
Label	dataLabel;

Method :

DisplayDialog(mainClass, String);
boolean handleEvent(Event);
Panel setButton( );
void makeDataField( );
void updataData( );
void makeLabel( );
void addFormComponent(GridBagLayout, Component, GridBagConstraints);
Color getBackColor( );
void displayDataField( );

Figure 4.3.1.11: *Descriptions of class DisplayDialog.*



class DisplaySearchDialog

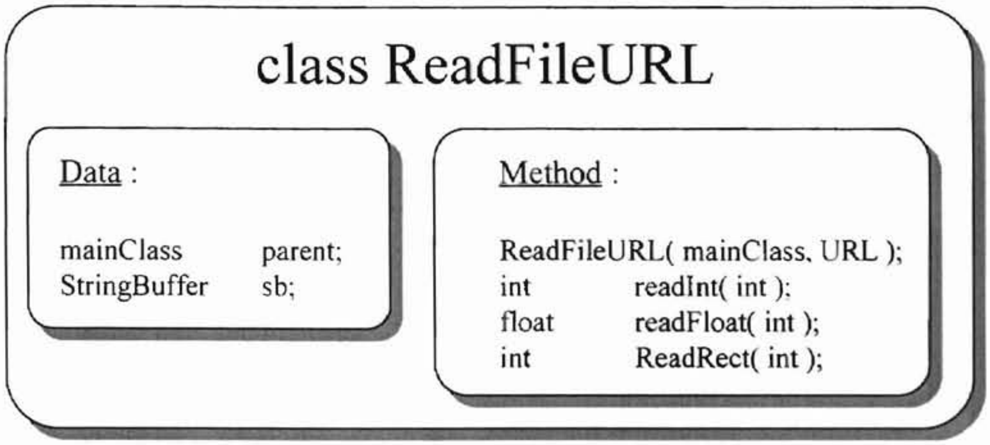
Data :

mainClass	parent;
TextField	rectField[ ][ ];
TextField	dataField;
Label	rectLabel;
Label	dataLabel;

Method :

DisplaySearchDialog(mainClass);
boolean handleEvent(Event);
Panel setButton( );
void makeDataField( );
void makeLabel( );
void addFormComponent(GridBagLayout, Component, GridBagConstraints);
void displayDataField( );

Figure 4.3.1.12: *Descriptions of class DisplaySearchDialog.*



class ReadFileURL

Data :

```
mainClass       parent;
StringBuffer    sb;
```

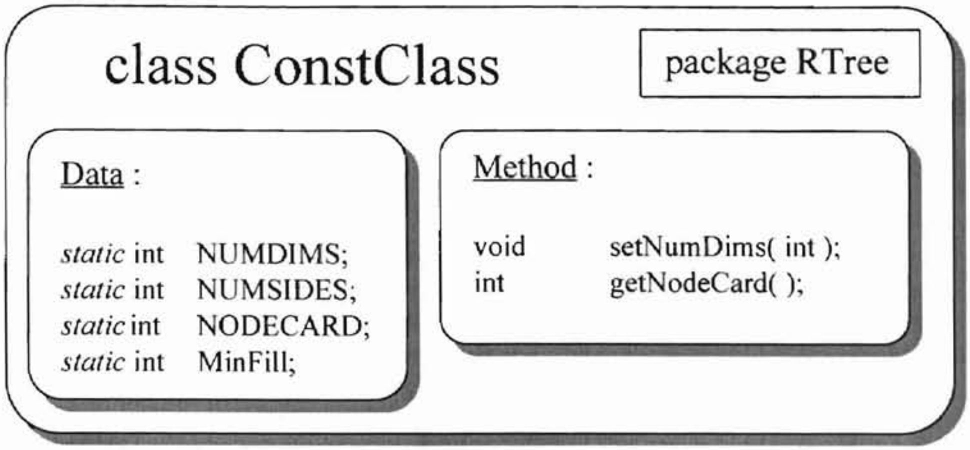
Method :

```
ReadFileURL( mainClass, URL );
int         readInt( int );
float       readFloat( int );
int         ReadRect( int );
```

Figure 4.3.1.13: *Descriptions of class ReadFileURL.*

4.3.2 All classes in RTree package

1. class ConstClass	
2. class BranchClass	
3. class RootClass	
4. class RectClass	extends ConstClass
5. class NodeClass	extends ConstClass
6. class SplitClass	extends ConstClass
7. class IndexClass	extends ConstClass
8. class DisplayClass	extends Frame

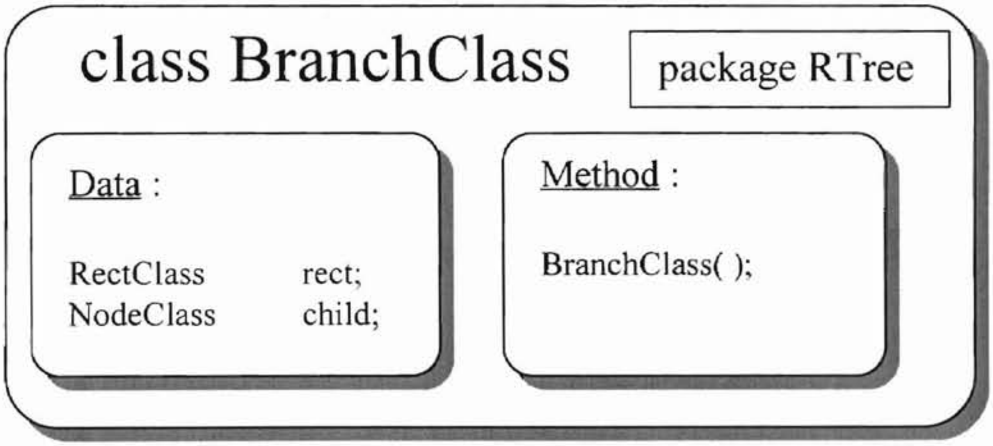


```
static int   NUMDIMS;
static int   NUMSIDES;
static int   NODECARD;
static int   MinFill;
```

```
void     setNumDims( int );
int      getNodeCard( );
```

Figure 4.3.2.1: *Descriptions of class ConstClass.*

"static" means this variable has only one copy in this program. NUMDIMS variable keeps the number of dimension that uses in this data structure. NUMSIDES variable is 2 * NUMDIMS. If we create 3-dimensional spatial data structure, the NUMDIMS is 3 and NUMSIDES is the total side of the rectangle (2 * NUMDIMS). NODECARD is the maximum number of entries in a node. MinFill is the minimum number of entries in a node except root.



```
RectClass     rect;
NodeClass     child;
```

```
BranchClass( );
```

Figure 4.3.2.2: *Descriptions of class BranchClass.*

"rect" is the variable of class RectClass. It contains an n-dimensional rectangle that covers all the rectangles of the child node. "child" variable contains the address of child node.

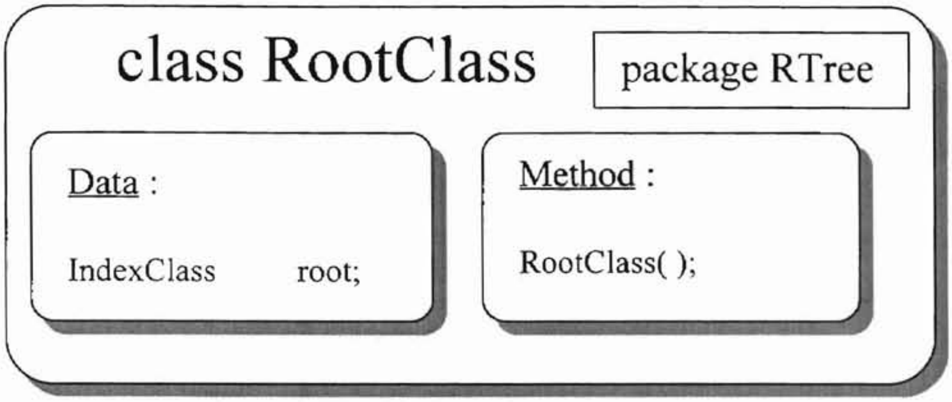


Figure 4.3.2.3: *Descriptions of class RootClass.*

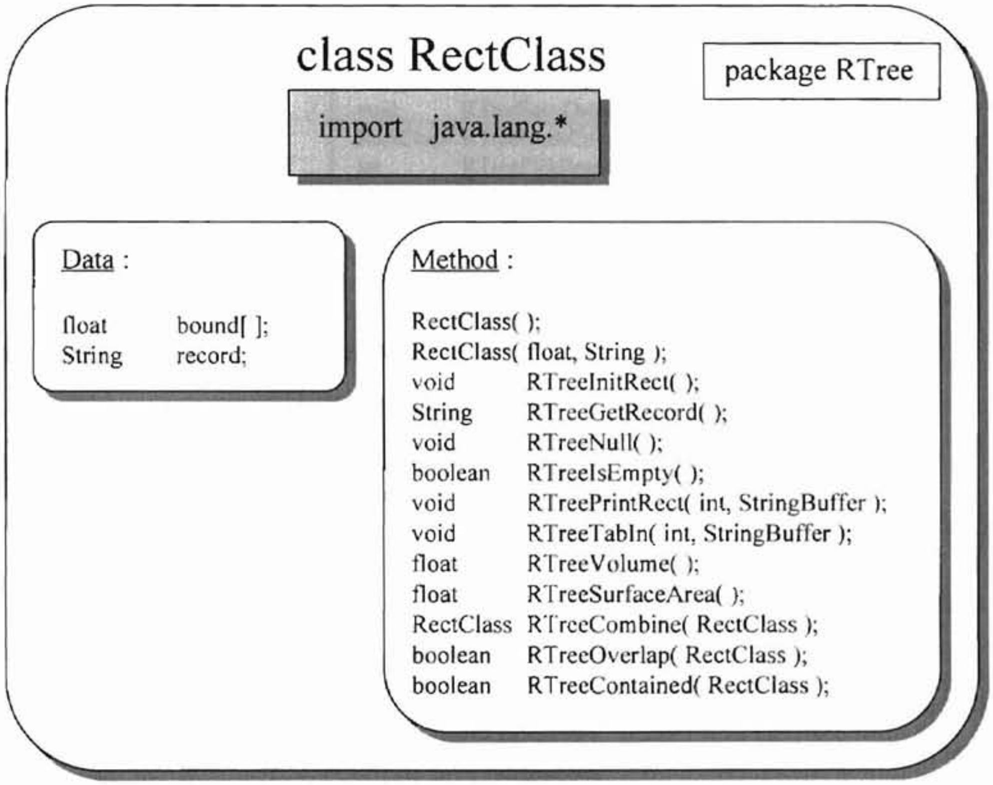
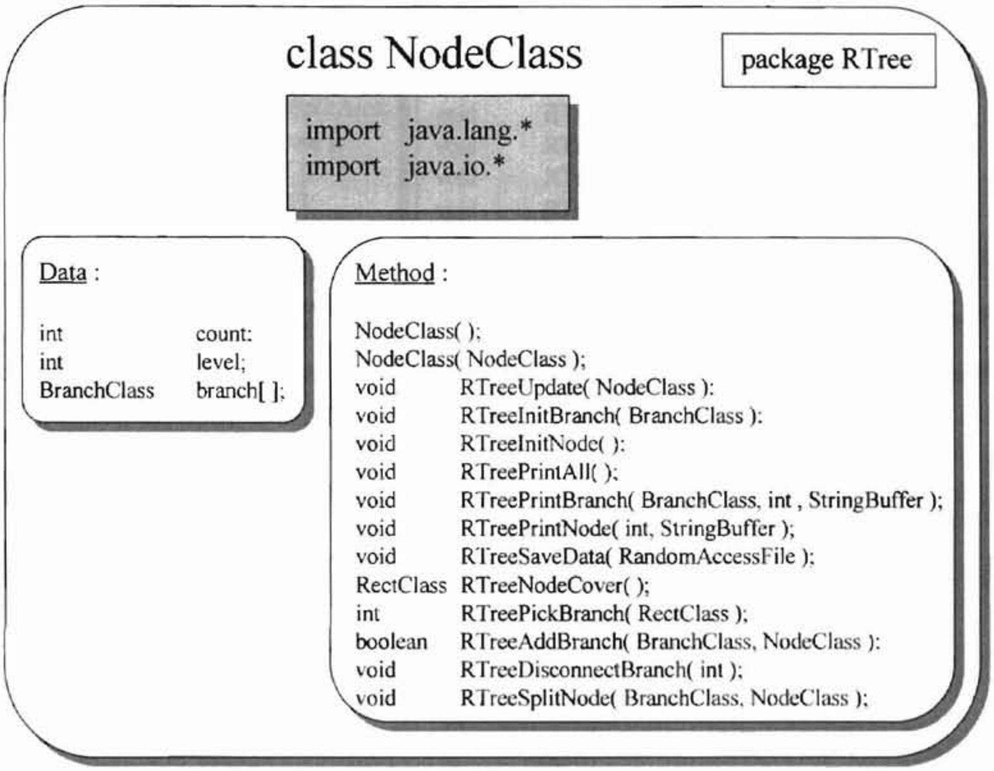

class RectClass
package RTree
import java.lang.*
Data :
float bound[ ];
String record;
Method :
RectClass( );
RectClass(float, String);
void RTreeInitRect( );
String RTreeGetRecord( );
void RTreeNull( );
boolean RTreeIsEmpty( );
void RTreePrintRect(int, StringBuffer);
void RTreeTabIn(int, StringBuffer);
float RTreeVolume( );
float RTreeSurfaceArea( );
RectClass RTreeCombine(RectClass);
boolean RTreeOverlap(RectClass);
boolean RTreeContained(RectClass);

Figure 4.3.2.4: *Descriptions of class RectClass.*

"bound" variable is an array of float values. The size of the array is equal to NUMSIDES. "record" keeps the object type. It will point to null if this node is a non-leaf node. In this program the object type is String value, but we can change this object type to another type, like image, sound, etc.



class NodeClass

package RTree

```
import   java.lang.*
import   java.io.*
```

Data :

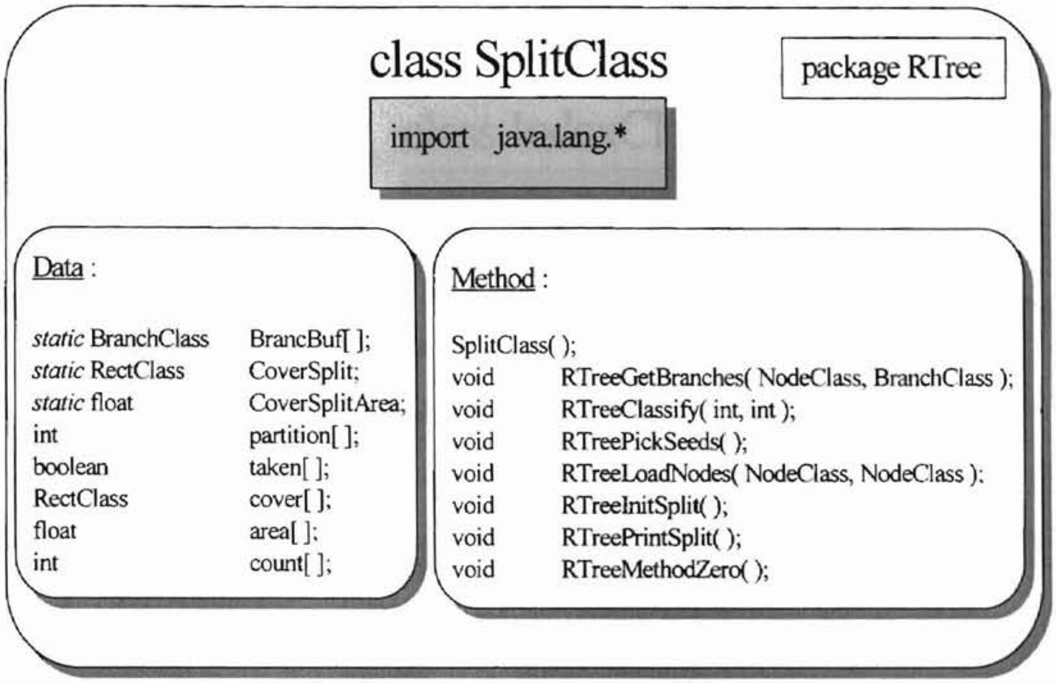
```
int             count:
int             level;
BranchClass     branch[ ];
```

Method :

```
NodeClass( );
NodeClass( NodeClass );
void        RTreeUpdate( NodeClass ):
void        RTreeInitBranch( BranchClass ):
void        RTreeInitNode( ):
void        RTreePrintAll( );
void        RTreePrintBranch( BranchClass, int , StringBuffer );
void        RTreePrintNode( int, StringBuffer );
void        RTreeSaveData( RandomAccessFile );
RectClass   RTreeNodeCover( );
int         RTreePickBranch( RectClass );
boolean     RTreeAddBranch( BranchClass, NodeClass ):
void        RTreeDisconnectBranch( int );
void        RTreeSplitNode( BranchClass, NodeClass );
```

Figure 4.3.2.5: *Descriptions of class NodeClass.*

"count" is the number of the child node. "level" is the level of this node. "branch" is the array of BranchClass. It contains all the properties of child nodes. This class uses for display the spatial data structure, and it has the function to split node when the value in count variable greater than NODECARD. This class also contains the algorithm PickNext, call RTreePickBranch member function, from the algorithm Quadratic Split [Guttman, 1994].



class SplitClass

package RTree

import java.lang.*

Data :

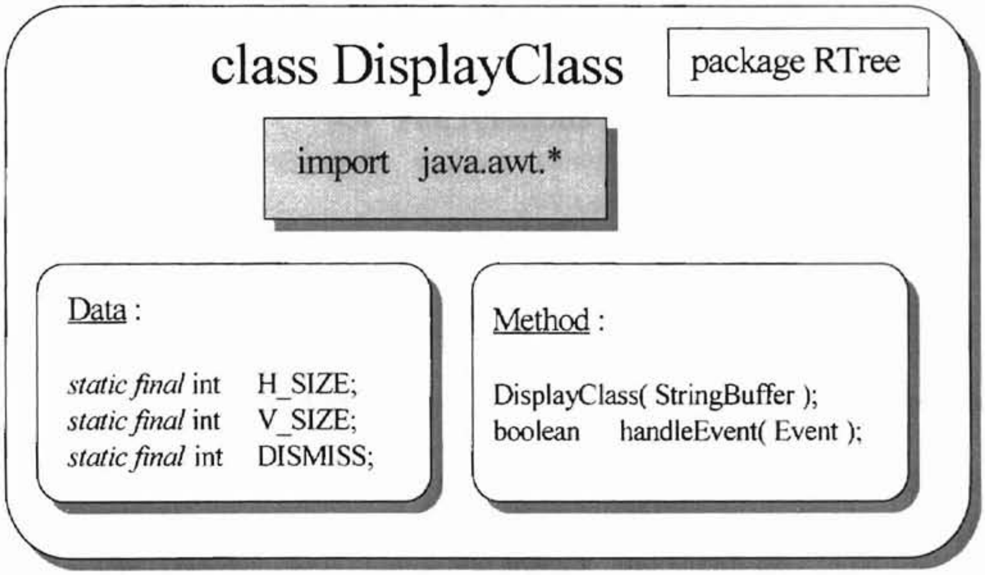
```
static BranchClass   BrancBuf[ ];
static RectClass     CoverSplit;
static float         CoverSplitArea;
int                  partition[ ];
boolean              taken[ ];
RectClass            cover[ ];
float                area[ ];
int                  count[ ];
```

Method :

```
SplitClass( );
void     RTreeGetBranches( NodeClass, BranchClass );
void     RTreeClassify( int, int );
void     RTreePickSeeds( );
void     RTreeLoadNodes( NodeClass, NodeClass );
void     RTreeInitSplit( );
void     RTreePrintSplit( );
void     RTreeMethodZero( );
```

Figure 4.3.2.6: *Descriptions of class SplitClass.*

We choose the algorithm Quadratic Split [Guttman, 1994] for handle split node. This algorithm calls two more algorithms: PickSeeds, call RTreePickSeed in this program, and PickNext, call RtreePickBranch in the class NodeClass.



class DisplayClass

package RTree

import java.awt.*

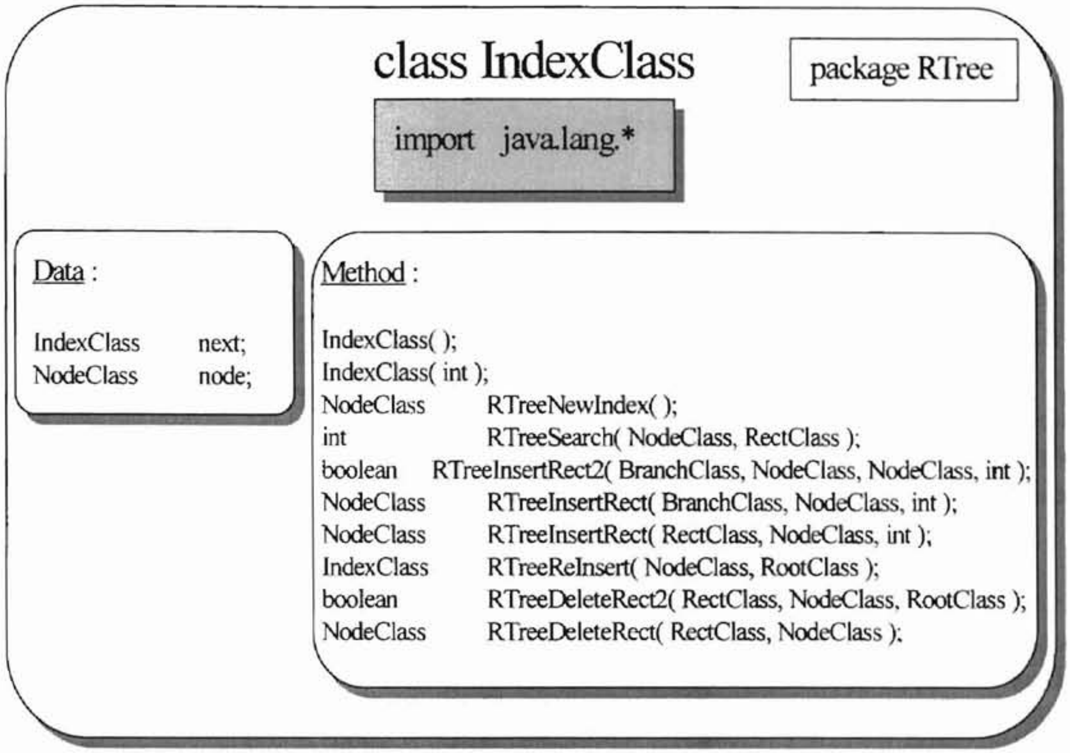
Data :

```
static final int   H_SIZE;
static final int   V_SIZE;
static final int   DISMISS;
```

Method :

```
DisplayClass( StringBuffer );
boolean     handleEvent( Event );
```

Figure 4.3.2.7: *Descriptions of class DisplayClass.*



```
class IndexClass                    package RTree

import  java.lang.*

Data :

IndexClass      next;
NodeClass       node;

Method :

IndexClass( );
IndexClass( int );
NodeClass       RTreeNewIndex( );
int             RTreeSearch( NodeClass, RectClass );
boolean    RTreeInsertRect2( BranchClass, NodeClass, NodeClass, int );
NodeClass       RTreeInsertRect( BranchClass, NodeClass, int );
NodeClass       RTreeInsertRect( RectClass, NodeClass, int );
IndexClass      RTreeReInsert( NodeClass, RootClass );
boolean         RTreeDeleteRect2( RectClass, NodeClass, RootClass );
NodeClass       RTreeDeleteRect( RectClass, NodeClass );
```

Figure 4.3.2.8: *Descriptions of class IndexClass.*

This class contains the insert, delete, and search member functions. Insert and delete member functions are implemented from algorithm in [Guttman, 1994].

4.4 The relations among classes

The processing of each class is based on communication among its objects as outlined below (see Figure 4.4.1)

1. The mainClass class is the main method. The program is started from this class. This class initializes all of the variables and create all of the dialog boxes
2. The ToolBar class creates the menus, and menu bar. The mainClass calls this class for showing all of the menus.

3. The StatusBar class shows the status line.
4. All dialog classes displays dialog boxes of each menu.
5. Package Rtree is the package that contains all of the spatial database index classes.

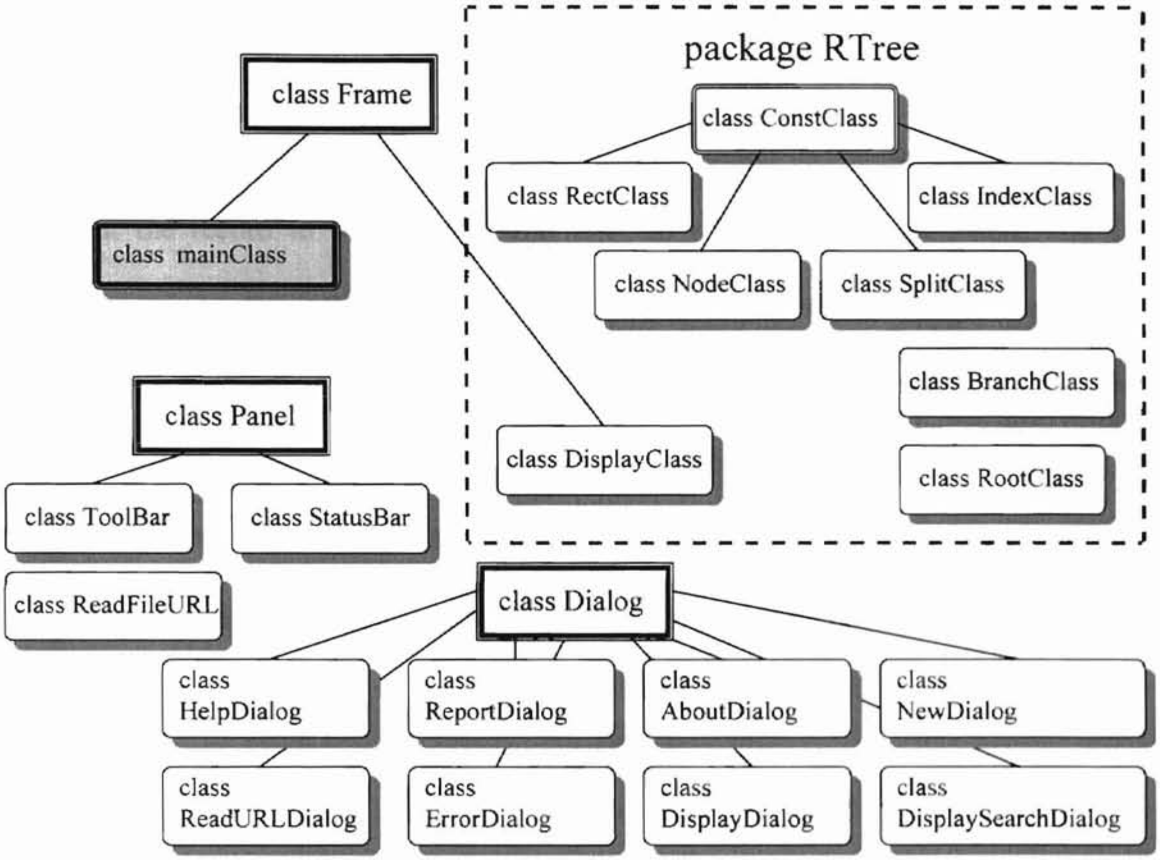


Figure 4.4.1: *The relations among classes.*

4.5 Description of data

We have four input data files for testing the program. The first two data files have 25 records with 2 and 3 dimensional spatial for testing the correctness of the program.

The last two data files have 1200 records with 3 and 4 dimensional spatial data respectively.

4.6 Data analysis

The first two data files are generated by the Java random function with range [0, 50]. The low side is guaranteed to be less than the high side. We draw the graph to find the relation of all records and write the separate Java codes to test each algorithm. First we input the record to the program and test the correctness of the insert, and split functions step by step. Second we try the different combinations of each function, for example insert, delete, insert, search, delete, ..., etc., and check the result every step with the programs that we write for checking each function. The last two data files are generated by the Java random function with maximum value of 50,000 and minimum value of 0. These input data files used for testing the program that can handle large input data files.

CHAPTER V

CONCLUSIONS, AND FUTURE WORK

In this paper, we have introduced the multi-dimensional spatial database application. The application was tested under Solaris 2.4 on a SPARC machine, but the application can run on PC under Window 95/Window NT, Macintosh under MacOS, and the machines that use these Operating System: OS/2, HPUX, AIX, SunOS, Solaris, UnixWare, Linux, NetWare 4.0, and MVS.

5.1 Conclusions

This thesis consists of two parts: the development of a multi-dimensional spatial database index system and the implementation of the user interface. In the first part we used the algorithm in [Guttman 1984], and added the function to implement multi-dimensional data object and to created simple spatial database. In the second part we added menu-driven and graphical Input/Output. The application performs both querying and graphical representations of spatial data. Existing structures can perform operations like search, insert, and delete. This application can be used for various applications such as Geographic Information System (GIS), Very Large-Scale Integrated (VLSI) design, and Computer-Aided Design (CAD) by changing the type of the spatial object.

5.2 Future Work

The following are some of the areas where future work on this application is suggested:

1. Linking the other format of the database file to the application would greatly enhance the usability of the application.
2. Using more graphical user interface in part of display output and other parts. This allow the users to easily use the program.
3. In this application the local site cannot update data file on the remote site directly because of the security of the Internet. There are a few ways to update a data file on the remote site from the local site. One of the concepts is sending the data to the Common Gateway Interface (CGI) on the remote site from the local site, but the remote site must have the script command to receive the data in this CGI form.

REFERENCES

1. [Aitken, 1996] Aitken, G. Moving from C++ to Java: *Dr. Dobb's Journal.* March 1996.
2. [Arnold and Gosling, 1996] Arnold, K., and Gosling, J. *The Java Programming Language.* Addison-Wesley, Reading, Massachusetts, 1996.
3. [Bang, 1995] Bang, K.S. *New Efficient Spatial Index Structures for Spatial Databases, PML and SMR.* Oklahoma State University, Department of Computer Science, September, 1995.
4. [Bentley, 1975] Bentley J.L. Multidimensional binary search trees for associative searching: *Communications of the ACM.* 18(9): 509-517; 1975.
5. [Campione and Walrath, 1996] Campione, M., and Walrath, K. *The Java Language Tutorial: Object-Oriented Programming for the Internet.* Addison-Wesley, Reading, Massachusetts, 1996.
6. [Daniel, 1994] Daniel, G. *UNIX in a nutshell : System V edition*, 2nd edition. O'Reilly & Associates, Sebastopol, California, August 1994.
7. [December, 1995] December, J. *Presenting Java.* Sams.net, Indianapolis, IN, 1995.
8. [Deitel, 1990] Deitel, H.M. *Operating Systems*, 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

9. [Eckel, 1996] Eckel, B. *Thinking in Java.* Prentice-Hall, Englewood Cliffs, New Jersey, June 1996.

10. [Ellis and Bjarne, 1992] Ellis, M., and Bjarne, S. *The Annotated C++ Reference Manual.* Addison-Wesley, Reading, Massachusetts, 1992.

11. [Flanagan, 1996] Flanagan, D. *Java in a Nutshell.* O'Reilly & Associates, Sebastopol, California, 1996.

12. [Gosling, Joy, and Steele, 1996] Gosling, J., Joy, B., and Steele, G. *The Java language Specification.* Addison-Wesley, Reading, Massachusetts, 1996.

13. [Gosling and McGilton, 1995] Gosling, J., and McGilton, H. *The Java Language Environment.* Sun Microsystems Computer Company, Mountain View, California, October 30, 1995.

14. [Guttman, 1984] Guttman, A. R-trees: a dynamic index structure of spatial searching: *Proceedings of the SIGMOD Conference.* Boston: pages 47-57; 1984.

15 [Hoare, 1973] Hoare, C.A.R. *Hints on Programming Language Design.* Stanford University Computer Science Department, Technical Report NO CS-73-403, December 1973.

16. [Hoelt and Samet, 1992] Hoelt, E.G. and Samet, H. A Qualitative Comparison Study of Data Structures for Large Line Segment Databases: *Proceedings of ACM SIGMOD,* 205-214; 1992.

17. [Hoff, Shaio, and Starbuck, 1996] Hoff, A.V., Shaio, S., and Starbuck, O. *Hooked on JAVA.* Addison-Wesley Publishing Company, Reading, Massachusetts. 1996.

18. [Hughes, 1994] Hughes, K. *Entering the World-Wide Web: A Guide to Cyberspace.* http://www.eit.com/web, May 1994.

19. [Jackson, 1996] Jackson, J.R. *Java by example.* SunSoft Press, Mountain View, California, 1996.

20. [Kamel and Faloutsos, 1992] Kamel, I., and Faloutsos, C. Parallel R-tree: *SIGMOD.* 195-204; 1992.

21. [Kamel and Faloutsos, 1994] Kamel, I., and Faloutsos, C. *Hilbert R-tree: An Improved R-tree using fractals*. University of Maryland, College Park, Feb. 1994.

22. [Kernigkan and Dennis, 1988] Kernighan, Brian W., and Dennis, M.R. *The C Programming Language,* 2nd edition. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

23. [Lindholm and Yellin, 1996] Lindholm, T., and Yellin, F. *The Java Virtual Machine Specification.* Addison-Wesley, Reading, Massachusetts, 1996.

24. [Madsen, Birger, and Kristen, 1993] Madsen, O.L., Birger, M.P., and Kristen, N. *Object-Oriented Programming in the Beta Programming Language.* Addison-Wesley, Reading, Massachusetts, 1993.

25. [Matsuyama, Hao, and Nagao, 1984] Matsuyama, T., Hao, L.V., and Nagao, M. A file organization for geographic information systems based on spatial proximity: *Computer Vision, Graphic and Image Processing,* 26(3): 266-271; 1984.

26. [Naughton, 1996] Naughton, P. *The Java handbook.* Osborne McGraw-Hill, Berkeley, California, 1996

27. [Perkins, 1995] Perkins C.L. http://rendezvous.com/java, 1995.

28. [Pfaffenberger, 1995] Pfaffenberger, B. *Netscape Navigator: Surfing the Web and Exploring the Internet.* AP Professional, Chestnut Hill, Massachusetts, 1995.

29. [Ritchey, 1995] Ritchey, T. *Java!.* New Riders, Indianapolis, IN, 1995.

30. [Robinson, 1984] Robinson, J.T. The K-D-B-tree: A search structure for large multidimensional dynamic indexes: *Proceedings of ACM SIGMOD Conference on Management of Data.* 1984.

31. [Southerton and Perkins 1994] Southerton, A., and Perkins, E.C. Jr. *The UNIX and X COMMAND COMPENDIUM: A Dictionary for High-Level Computing.* John Wiley & Sons, Inc, New York, New York, 1994.

32. [Stroustrup, 1994] Stroustrup, B. *The C++ Programming Language,* 2nd edition. Addison-Wesley, Reading, Massachusetts, 1994.

33. [Theodoridis and Sellis, 1994] Theodoridis, Y., and Sellis, T. Optimization Issues in R-tree Construction: *IGIS,* 270-273; 1994.

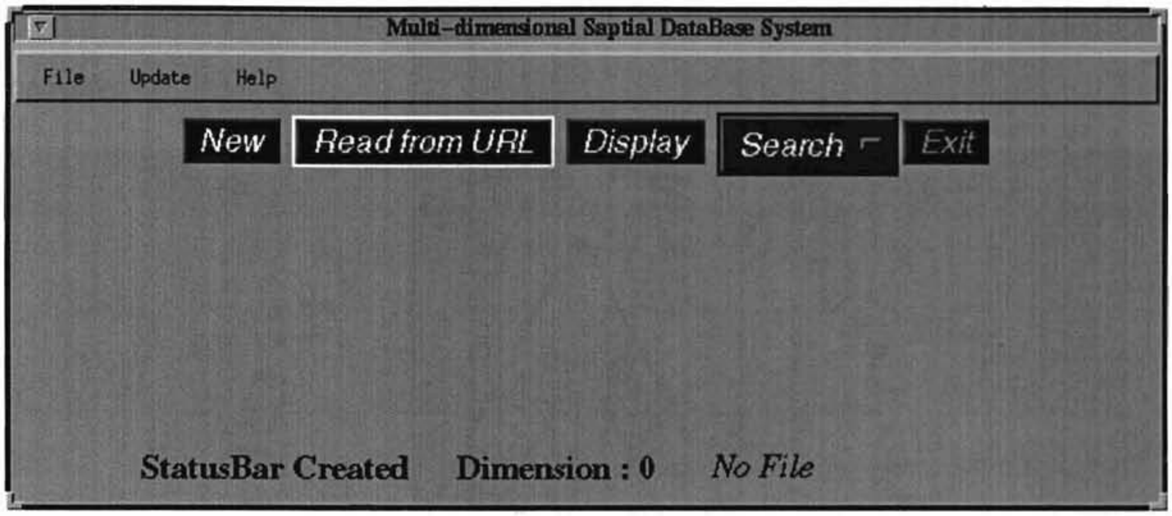
APPENDIXES

APPENDIX A

USER MANUAL

USER MANUAL

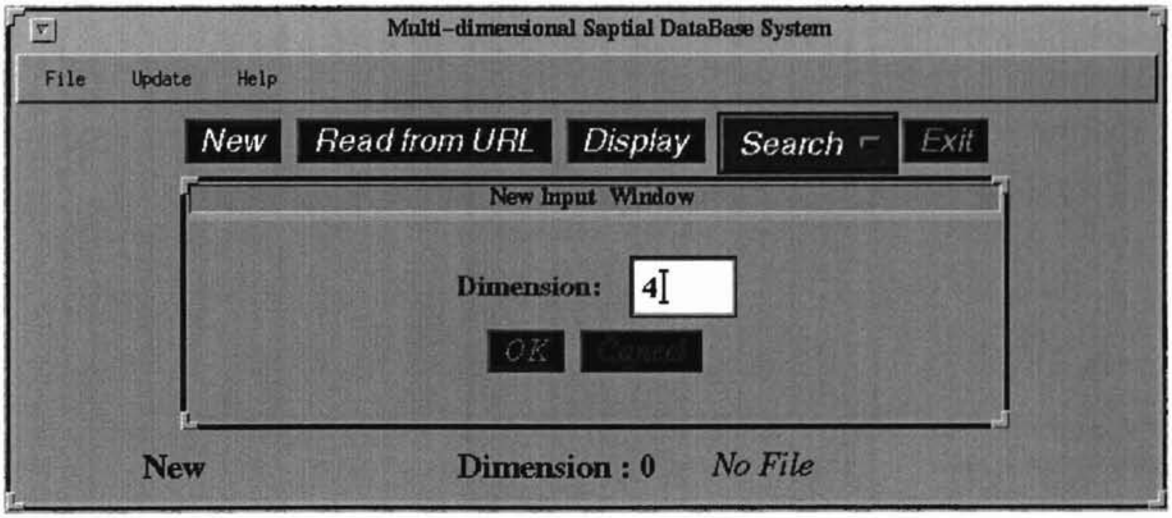
The main window has five push buttons, and three menu bars (see Figure A-1).



StatusBar Created Dimension : 0 *No File*

Figure A-1: *Main window of the application.*

When the "*New*" button is pushed the "New Input Window" dialog box is showed (see Figure A-2). The input in field dimension should be an integer greater than 2. After input dimension field the button "*OK*" is for created the new multi-dimensional spatial data structure with the given dimension number, or "*Cancel*" for cancel.

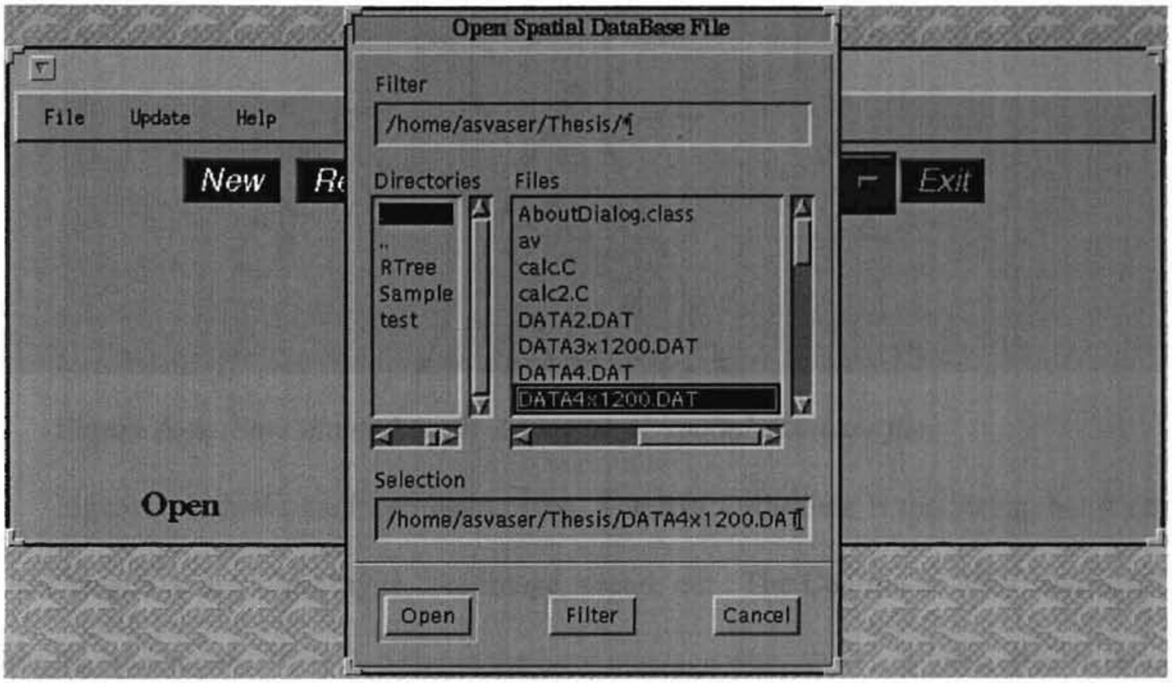


Dimension: 4

New Dimension : 0 *No File*

Figure A-2: *New window dialog for created new spatial data structure.*

When the users choose "*Open*", or "*Save*" from menu bar "*File*", the program will show Figure A-3, and A-4 respectively for load and save the spatial database file to the local site. These two menus can not run via on the Internet because of the security of the network.



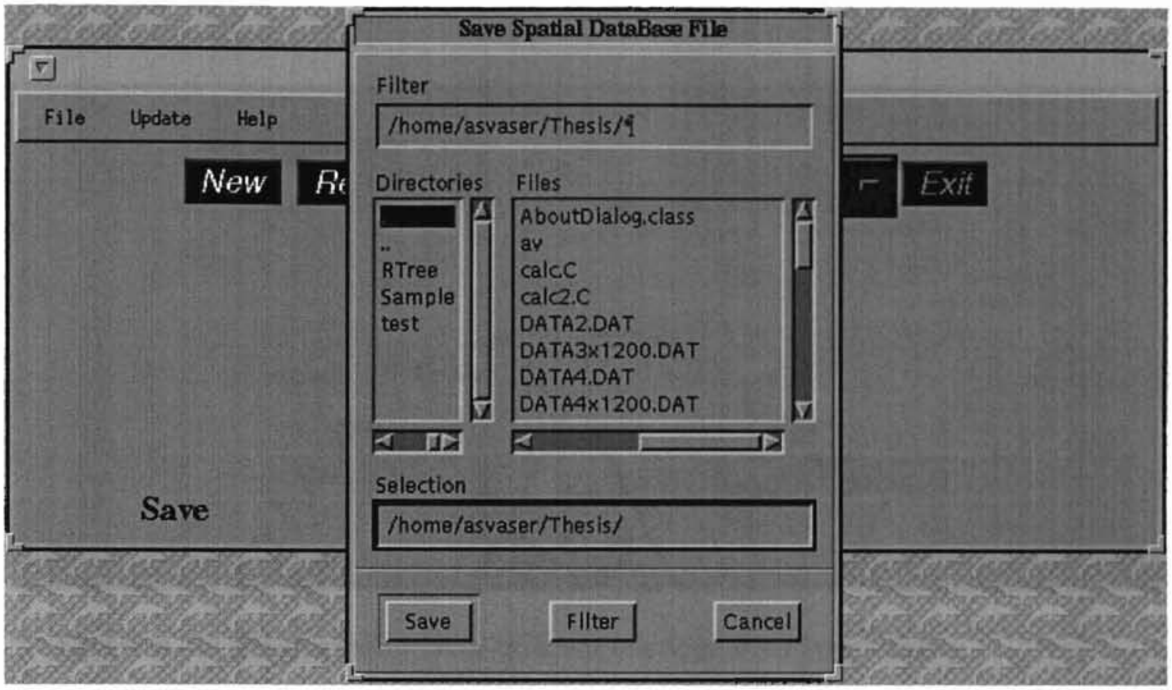
Filter
/home/asvaser/Thesis/*

Directories
..
RTree
Sample
test

Files
AboutDialog.class
av
calc.C
calc2.C
DATA2.DAT
DATA3x1200.DAT
DATA4.DAT
DATA4x1200.DAT

Selection
/home/asvaser/Thesis/DATA4x1200.DAT

Figure A-3: *Open dialog box for loading local spatial database file.*



Save Spatial DataBase File

Filter

/home/asvaser/Thesis/*

Directories

..
RTree
Sample
test

Files

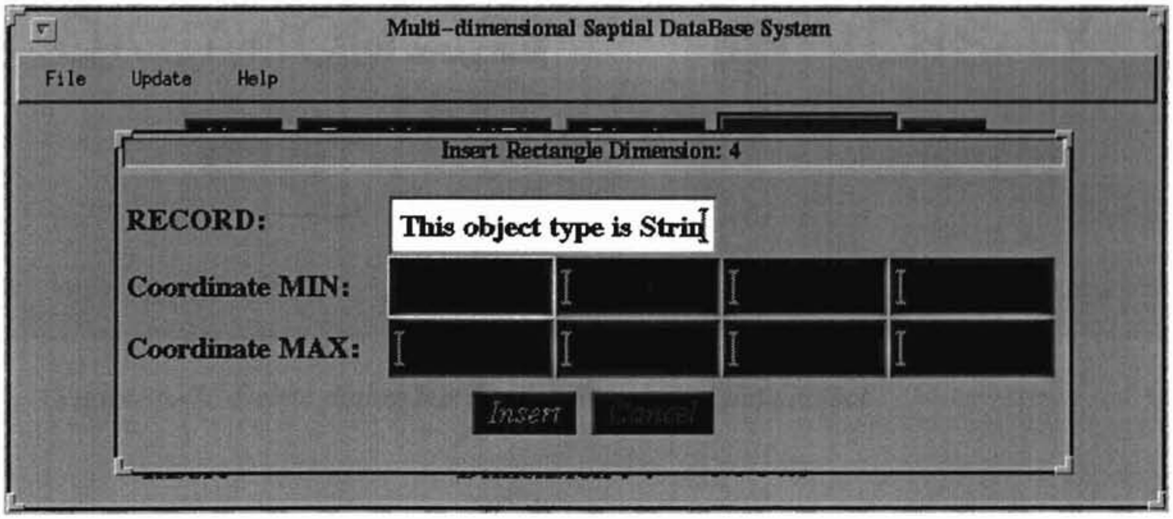
AboutDialog.class
av
calc.C
calc2.C
DATA2.DAT
DATA3x1200.DAT
DATA4.DAT
DATA4x1200.DAT

Selection

/home/asvaser/Thesis/

Figure A-4: *Save dialog box for saving local spatial database file.*

Figure A-5 shows the insert dialog box. The RECORD field is the String, but it can be changed to any object type, like image, sound, etc. The Coordinate MIN, and MAX are the float number. Fig A-6 shows the error message when there are some error in input field.



Multi-dimensional Saptial DataBase System

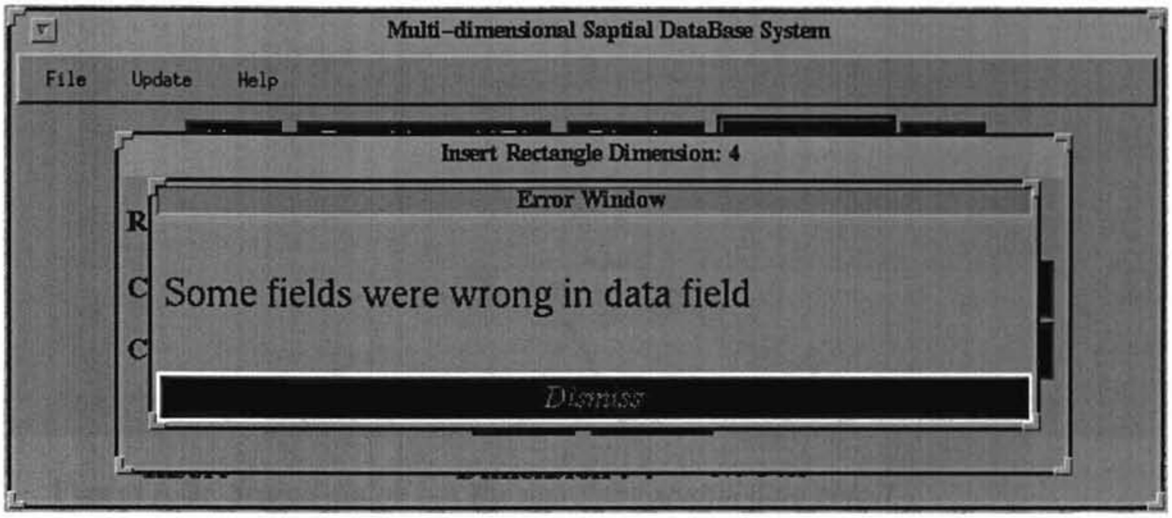
Insert Rectangle Dimension: 4

RECORD: This object type is Strin

Coordinate MIN:

Coordinate MAX:

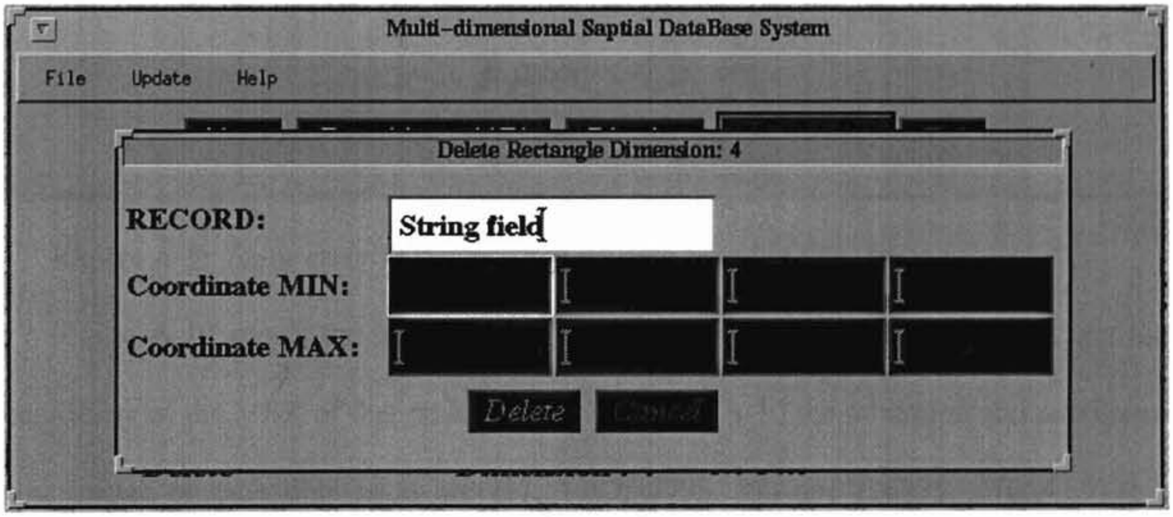
Figure A-5: *Insert dialog box for insertion new spatial data object.*


Multi-dimensional Saptial DataBase System
File
Update
Help
Insert Rectangle Dimension: 4
Error Window
Dismiss

Some fields were wrong in data field

Figure A-6: *Error window show an error message from insert dialog.*

Figure A-7, and A-8 are the delete dialog box, and search dialog box respectively. The input field in the delete dialog is the same as the input in insert dialog, but in the search dialog there is no RECORD field.


Multi-dimensional Saptial DataBase System
File
Update
Help
Delete Rectangle Dimension: 4
Delete
Cancel

RECORD: String field

Coordinate MIN:

Coordinate MAX:

Figure A-7: *Delete dialog box for deletion spatial data object.*

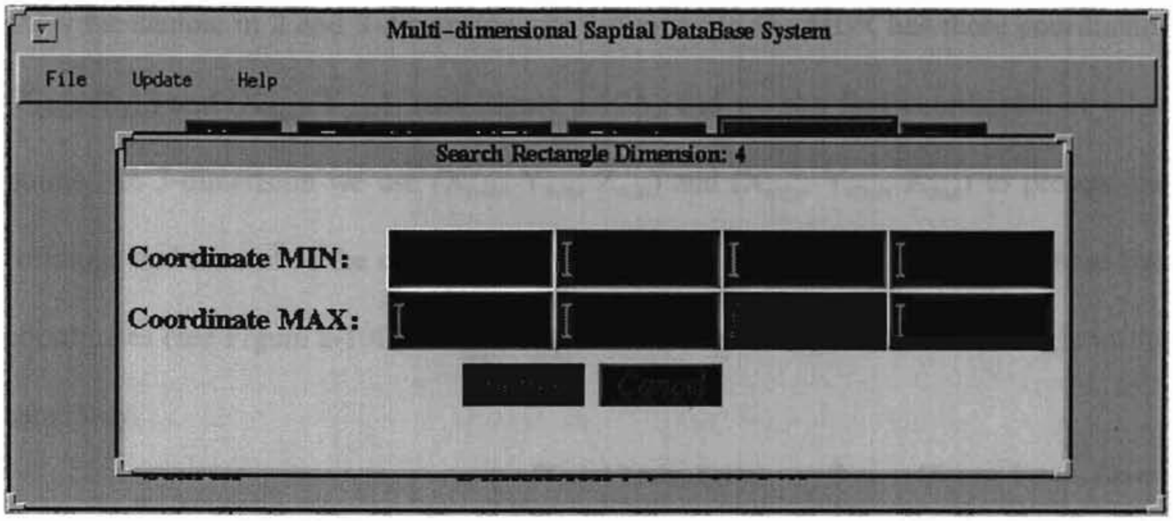
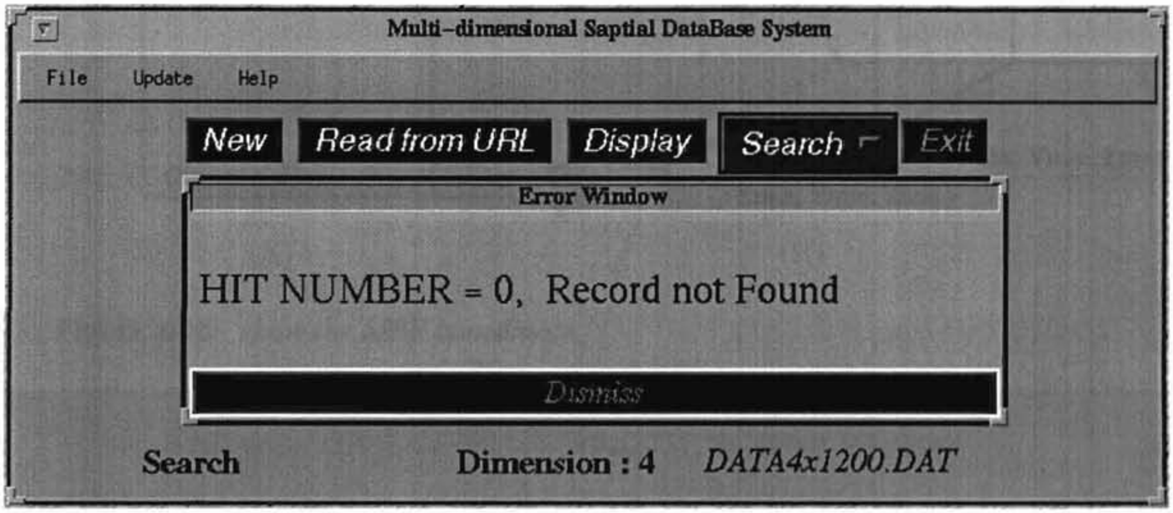

Multi-dimensional Saptial DataBase System
File
Update
Help
Search Rectangle Dimension: 4
Coordinate MIN:
Coordinate MAX:
Cancel

Figure A-8: *Search dialog box for searching spatial data object.*



HIT NUMBER = 0, Record not Found

Search Dimension : 4 DATA4x1200.DAT

Figure A-9: *Show error message when no data found in this structure.*

Figure A-10 shows the output of the spatial data objects. MIN and MAX are the coordinate of the MBR of that node. From the Figure A-10 the minimum and maximum coordinates of each dimension are (17, 1697, 7795, 39) and (26295, 37054, 37362, 33120) in a 4-dimensional space. In this program we keep only two coordinates to present the MBR, and these two coordinates must be the minimum and maximum coordinates. Because we can not draw the rectangle in more than 3-dimension, we will show the sample in 2 and 3-dimension. In 2-dimension the MBR has these coordinates (X_{min}, Y_{min}) and (X_{max}, Y_{max}) (see Figure a-10A) and we can find coordinates of other points. In 3-dimension we use (X_{min}, Y_{min}, Z_{min}) and (X_{max}, Y_{max}, Z_{max}) to present the rectangle and the rest of the coordinates of other points are the combination of these two coordinates (see Figure a-10B). For the n-dimension we can present the rectangle in the same way.

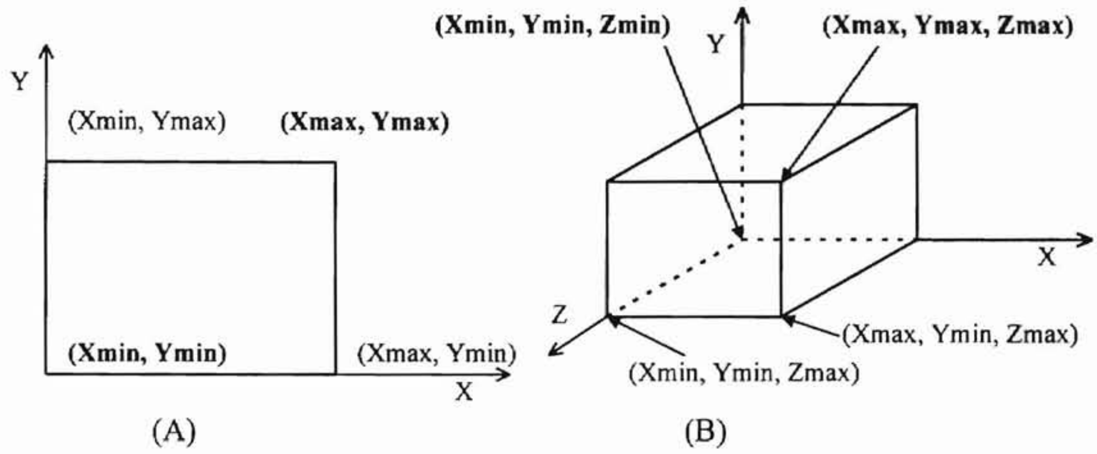
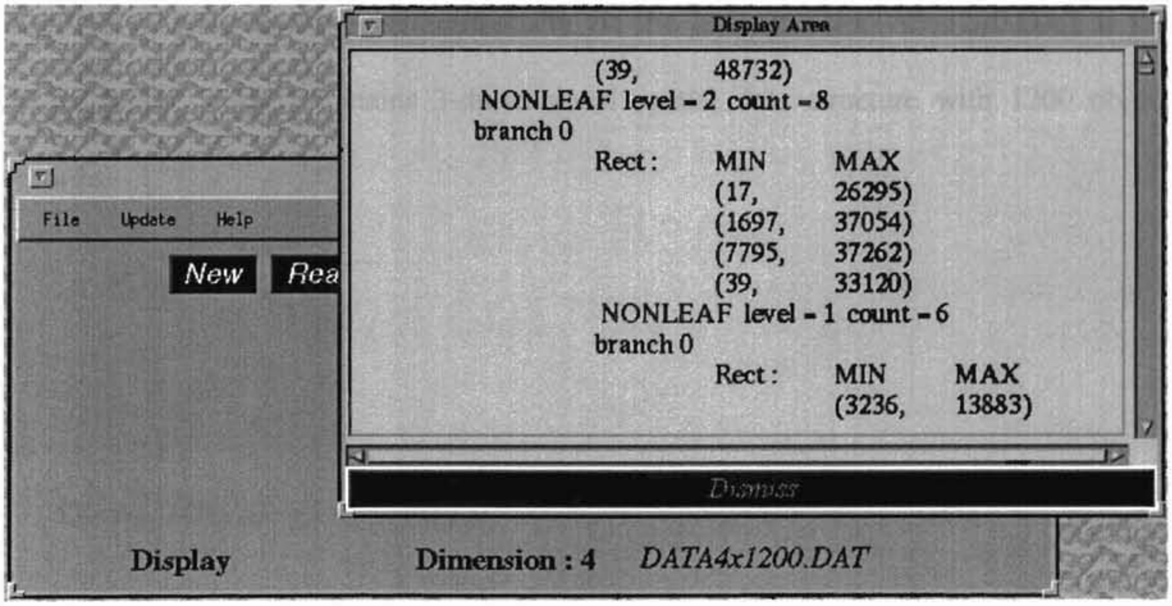


Figure a-10: *show the MBR coordinate.*

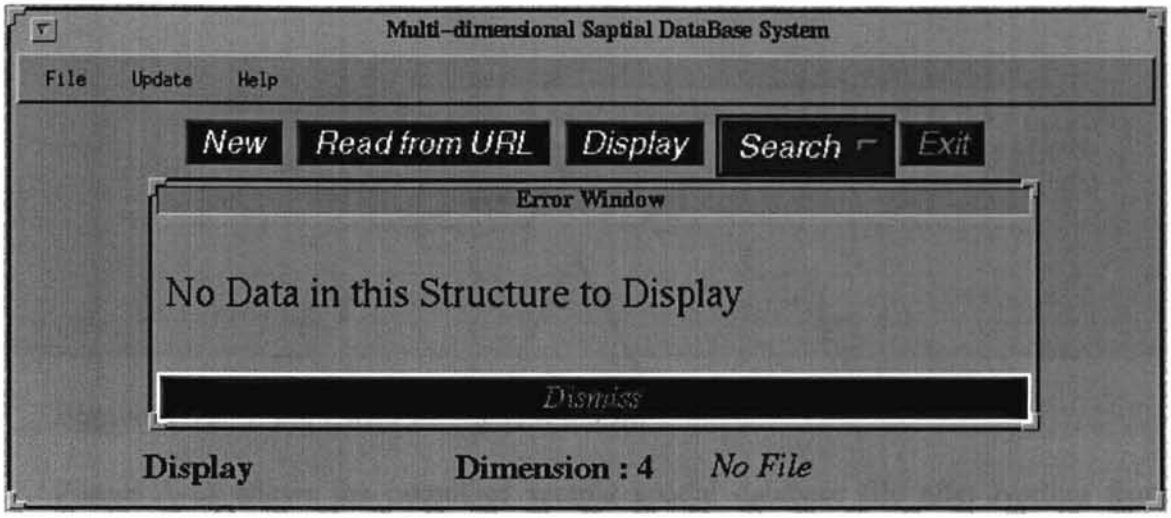


```
        (39,        48732)
    NONLEAF level = 2 count = 8
    branch 0
        Rect :      MIN         MAX
                    (17,        26295)
                    (1697,      37054)
                    (7795,      37262)
                    (39,        33120)
        NONLEAF level = 1 count = 6
        branch 0
                    Rect :      MIN         MAX
                                (3236,      13883)
```

Dimension : 4   *DATA4x1200.DAT*

Figure A-10: *Display all of the spatial data object.*

Figure A-11 shows an error message when the users try to display an empty spatial data structure

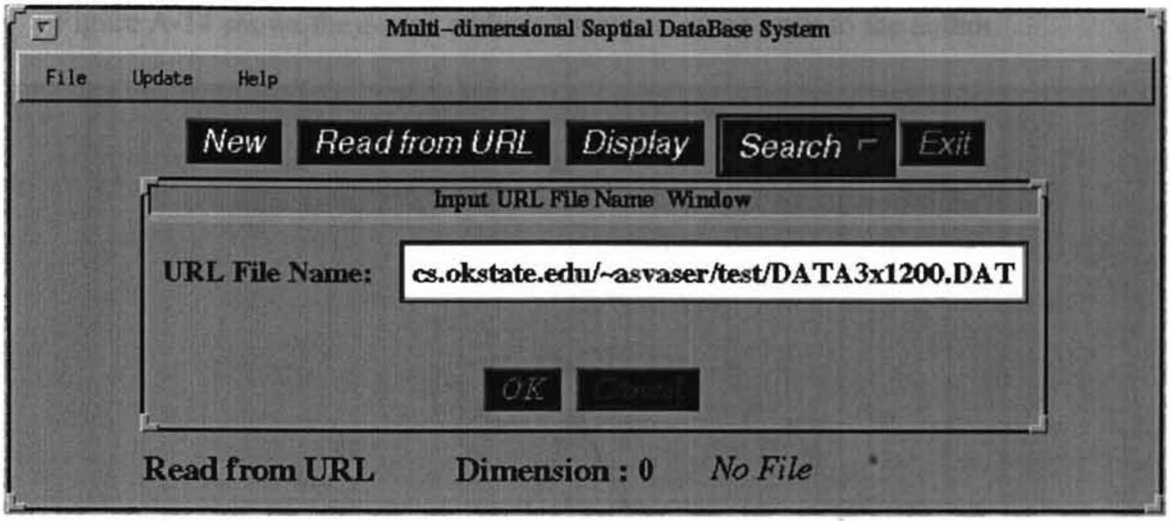


No Data in this Structure to Display

Display Dimension : 4 *No File*

Figure A-11: *Show an error message when user try to display an empty structure.*

Figure A-12 shows the read database file from URL (Universal Resource Location) dialog. The "URL File Name" field is the site that user want to get the remote spatial database file. From the Figure A-12 we read the database file from http://www.cs.okstate.edu/~asvaser/test site via the Internet. DATA3x1200.DAT is the database file , and it contains 3-dimensional spatial data structure with 1200 object records.

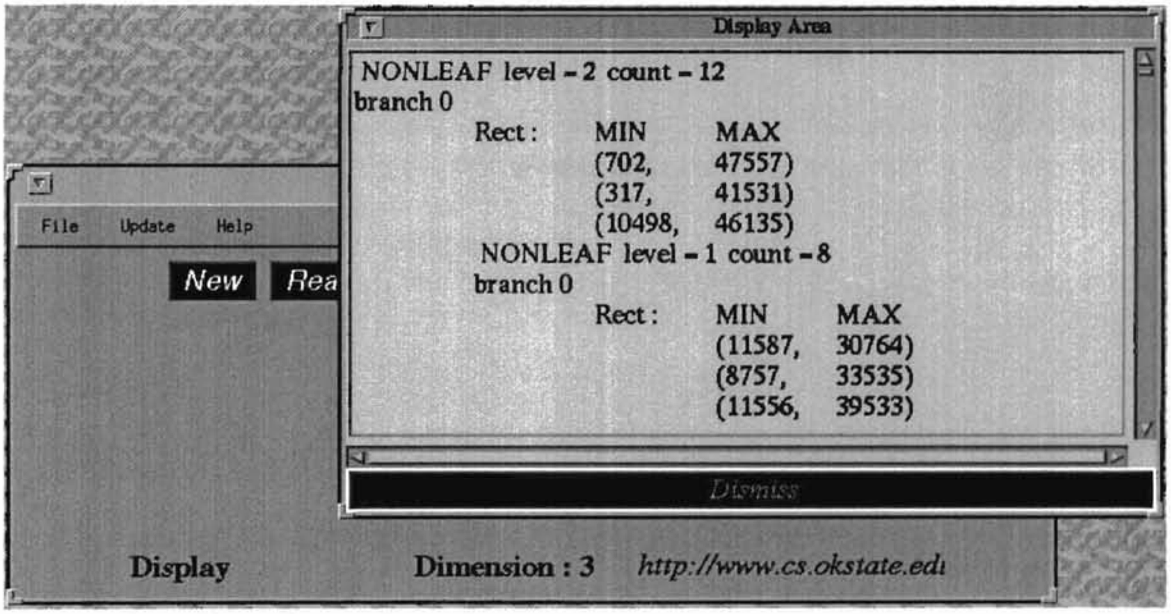

Multi-dimensional Saptial DataBase System
File
Update
Help
New
Read from URL
Display
Search
Exit
Input URL File Name Window
OK
Cancel

URL File Name: cs.okstate.edu/~asvaser/test/DATA3x1200.DAT

Read from URL   Dimension : 0   *No File*

Figure A-12: *Input URL File Name Window.*

Figure A-13 shows the output of sample spatial database file after reading from remote site. This display dialog shows the root in this data structure. This data structure has 2 level and the root contains 12 pointers to child node.

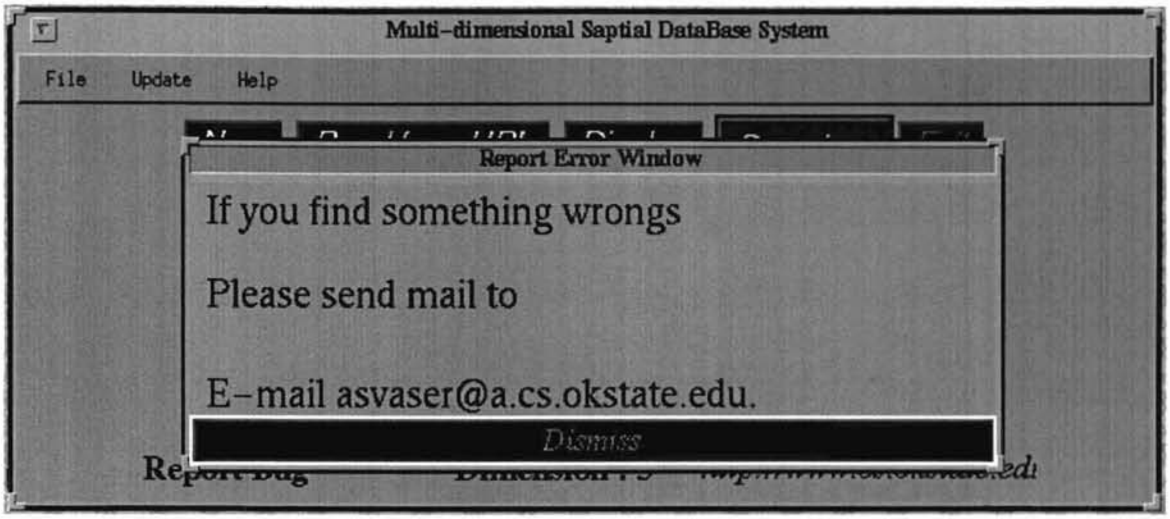


```
NONLEAF level - 2 count - 12
branch 0
          Rect:    MIN       MAX
                   (702,     47557)
                   (317,     41531)
                   (10498,   46135)
          NONLEAF level - 1 count - 8
          branch 0
                   Rect:     MIN       MAX
                             (11587,   30764)
                             (8757,    33535)
                             (11556,   39533)
```

Display   Dimension : 3   *http://www.cs.okstate.edu*

Figure A-13: *Display the output from the remote spatial database file.*

Figure A-14 shows the e-mail address for report some error to the author.



If you find something wrongs

Please send mail to

E-mail asvaser@a.cs.okstate.edu.

Figure A-14: *E-mail address for report some error.*

APPENDIX B

PROGRAM LISTING

```
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
/*                                                                    */
/*              Multi-dimensional Spatial Database System             */
/*                                                                    */
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
/*                                                                    */
/*      Author   : Veera Asvasermcharoen.                             */
/*      Date     : July, 1996                                         */
/*      course   : COMSC 5000 - Thesis                                */
/*      Advisor  : Dr. Huizhu Lu.                                     */
/*                                                                    */
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
/*     This application is a multi-dimensional spatial database system.  The   */
/* application is written using an object-oriented programming language ( the Java  */
/* language)                                                          */
/*     The application performs both querying and representations of spatial data.  */
/* This application can be used for various  Geographic Information System (GIS),  */
/* Very Large-Scale Integrated (VLSI) design, and Computer-Aided Design (CAD)  */
/* by changing the type of the spatial object.                       */
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
//
//      AUTHOR                  : VEERA ASVASERMCHAROEN
//      COURSE                  : COMSC 5000 - Thesis
//      TOPIC                   : Multi-dimensional spatial database system.
//      PROGRAM NAME            : ConstClass.java
//      PACKAGE FILE            : RTree
//      MACHINE                 : SPARC memory 64 Mb
//      OPERATING SYSTEM        : Solaris 2.4
//      LANGUAGE                : java language
//      COMPILER                : java compiler from Sun Microsystems version 1.0.2
//      DATE                    : July, 1996
//
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */

package    RTree;

/**
 *  Final member means this member cannot change
 */
public  class    ConstClass {
    final static int  PGSIZE = 1024;
    static int        NUMDIMS;
    static int        NUMSIDES;
    static int        NODECARD;
    static int        MinFill;            //⌈(NODECARD / 2)⌉;

    public  static void setNumDims( int dim ) {
        float         temp;

        NUMDIMS = dim;
        NUMSIDES = 2 * NUMDIMS;
        NODECARD = (int)(PGSIZE - (2 * 4)) / (NUMSIDES * 6));
                          // 4 is the sizeof integer and 6 is the sizeof float
        temp = (float)(NODECARD / 2);
        MinFill = (int)temp;
        if( temp - MinFill > 0.0 )
            MinFill++;
    }

    public  static int    getNodeCard( ) {
        return  NODECARD;
    }
};
```

```
/* * * * * * * * * * * * * * * * * * * * * * * */
//
//    AUTHOR              : VEERA ASVASERMCHAROEN
//    COURSE              : COMSC 5000 - Thesis
//    TOPIC               : Multi-dimensional spatial database system.
//    PROGRAM NAME        : BranchClass.java
//    PACKAGE FILE        : RTree
//    MACHINE             : SPARC memory 64 Mb
//    OPERATING SYSTEM    : Solaris 2.4
//    LANGUAGE            : java language
//    COMPILER            : java compiler from Sun Microsystems version 1.0.2
//    DATE                : July, 1996
//
/* * * * * * * * * * * * * * * * * * * * * * * */

package   RTree;

public class   BranchClass {
    RectClass      rect;
    NodeClass      child;

/**
 *  Constructor for init data member
 */
    BranchClass( ) {
        rect = new RectClass();
        child = null;
    }
};
```

```
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
//
//     AUTHOR               : VEERA ASVASERMCHAROEN
//     COURSE               : COMSC 5000 - Thesis
//     TOPIC                : Multi-dimensional spatial database system.
//     PROGRAM NAME         : RootClass.java
//     PACKAGE FILE         : RTree
//     MACHINE              : SPARC memory 64 Mb
//     OPERATING SYSTEM     : Solaris 2.4
//     LANGUAGE             : java language
//     COMPILER             : java compiler from Sun Microsystems version 1.0.2
//     DATE                 : July, 1996
//
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */

package   RTree;

public class   RootClass {
    IndexClass     root;

    RootClass( ) {
        root = null;
    }
};
```

```
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
//
//      AUTHOR                  : VEERA ASVASERMCHAROEN
//      COURSE                  : COMSC 5000 - Thesis
//      TOPIC                   : Multi-dimensional spatial database system.
//      PROGRAM NAME            : RectClass.java
//      PACKAGE FILE            : RTree
//      IMPORT FILE             : java.lang.*
//      MACHINE                 : SPARC memory 64 Mb
//      OPERATING SYSTEM        : Solaris 2.4
//      LANGUAGE                : java language
//      COMPILER                : java compiler from Sun Microsystems version 1.0.2
//      DATE                    : July. 1996
//
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */

package   RTree;

import java.lang.*;

public class   RectClass extends  ConstClass {
    float   bound[];        // Xmin, Ymin, ..., Xmax, Ymax, ...
    String  record;         // Instead with null if this Rect is not an leaf node

/**
 *  Constructor for init class Float
 */
    public RectClass( ) {
        bound = new float[NUMSIDES];
        for( int i = 0; i < NUMSIDES; i++ )
        bound[i] = (float)0.0;
        record = new String();
    }

/**
 *  Overload Constructor for init class Float
 */
    public RectClass( float rectField[], String recordField ) {
        bound = new float[NUMSIDES];
        for( int i = 0; i < rectField.length; i++ )
            bound[i] = rectField[i];
        record = new String( recordField );
    }

/**
```

```
 *  Initialize a rectangle to have all 0 coordinates.
 *  Method function to delete Rect
 */
    public  void    RTreeInitRect( ) {
        for( int i = 0; i < NUMSIDES; i++ )
            bound[i] = (float)0.0;
        record = null;                  // instead with null if use Object class
    }

/**
 *  Method function return data object
 */
    public  String  RTreeGetRecord( ) {
        return  record;
    }

/**
 *  Set the first low side is higher than its opposite side
 */
    public  void    RTreeNull( ) {
        bound = new float[NUMSIDES];        // create other static variables

        bound[0] = (float)1.0;
        bound[NUMDIMS] = (float)-1.0;
        for( int i = 0; i < NUMDIMS; i++ )
            bound[i] = bound[i + NUMDIMS] = (float)0.0;
    }

/**
 *  Method function to check empty Rect
 */
    public  boolean    RTreeIsEmpty( ) {
        for( int i = 0; i < NUMSIDES; i++ )
            if( bound[i] != (float)0.0 )
                return  false;                  // not empty
        return  true;               // empty
    }

/**
 *  Print out the data for a rectangle
 */
    public  void    RTreePrintRect(int depth, StringBuffer s) {
        RTreeTabIn(depth, s);
        s.append("Rect :" + "\n");
        for( int i = 0; i < NUMDIMS; i++ ) {
            RTreeTabIn(depth + 1, s);
            s.append("(" + bound[i] + ", " + bound[i + NUMDIMS] + ")\n");
        }
    }

/**
 *  Print tab space
 */
    public  void    RTreeTabIn(int depth, StringBuffer s) {
        for( int i = 0; i < depth; i++ )
            s.append("\t");
    }

/**
 *  Calculate the n-dimensional volume of a rectangle
 */
    public  float   RTreeVolume( ) {
        float    temp = (float)1.0;

        for(int i = 0; i < NUMDIMS; i++ ) {
            temp *= ( bound[i + NUMDIMS] - bound[i] );
        }
        return  temp;
    }

/**
 *  Calculate the n-dimensional surface area of a rectangle
 */
    public  float   RTreeSurfaceArea( ) {
        float    temp = (float)1.0;

        for( int i = 0; i < NUMDIMS; i++ ) {
            float    face_area = (float)1.0;

            for( int j = 0; j < NUMDIMS; j++ )
            if( i != j )
                face_area *= (bound[j + NUMDIMS] - bound[j]);
            temp += face_area;
        }
        return  (float)2.0 * temp;
    }

/**
 *  Combine two rectangle, make one that include both.
 */
```

```
    public  RectClass  RTreeCombine( RectClass R ) {
        RectClass  new_class = new RectClass();

        for( int i = 0; i < NUMDIMS; i++ ) {
            new_class.bound[i] = Math.min( bound[i], R.bound[i] );
            int  j = i + NUMDIMS;
            new_class.bound[j] = Math.max( bound[j], R.bound[j] );
        }
        return  new_class;
    }

/**
 *  Decide whether two Object overlap
 */
    public  boolean     RTreeOverlap( RectClass R ) {
        for( int i = 0; i < NUMDIMS; i++ ) {
            int  j = i + NUMDIMS;          // index for high sides
            if( bound[i] > R.bound[j]  ||  R.bound[i] > bound[j] )
                return  false;  // not Overlap
        }
        return  true;                    // Overlap
    }

/**
 *  Decide whether Object is contained in Object R
 */
    public  boolean     RTreeContained( RectClass R ) {
        for( int i = 0; i < NUMDIMS; i++ ) {
            int  j = i + NUMDIMS;     // index for high sides
            if( bound[i] > R.bound[i]  ||  bound[j] < R.bound[j] )
                return  false;        // not Contained
        }
        return  true;         // Contained
    }
};
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * */
//
//    AUTHOR              : VEERA ASVASERMCHAROEN
//    COURSE              : COMSC 5000 - Thesis
//    TOPIC               : Multi-dimensional spatial database system.
//    PROGRAM NAME        : NodeClass.java
//    PACKAGE FILE        : RTree
//    IMPORT FILE         : java.lang.*      java.io.*
//    MACHINE             : SPARC memory 64 Mb
//    OPERATING SYSTEM    : Solaris 2.4
//    LANGUAGE            : java language
//    COMPILER            : java compiler from Sun Microsystems version 1.0.2
//    DATE                : July, 1996
//
/* * * * * * * * * * * * * * * * * * * * * * * * */

package   RTree;

import java.lang.*;
import java.io.*;

public class   NodeClass    extends    ConstClass {
    int              count;
    int              level;        // 0 is leaf, others positive
    BranchClass      branch[] = new BranchClass[NODECARD];

/**
 *  Constructor for init NodeClass
 */
    public  NodeClass( ) {
        count = 0;
        level = -1;
        for( int i = 0; i < NODECARD; i++ )
            branch[i] = new BranchClass();
    }

/**
 *  Constructor for init NodeClass
 */
    public  NodeClass( NodeClass tmp ) {
        RTreeUpdate( tmp );
    }

/**
 *  Method function to copy the content of NodeClass
 */
    public void    RTreeUpdate( NodeClass tmp ) {
        count = tmp.count;
        level = tmp.level;
        for( int i = 0; i < NODECARD; i++ ) {
            branch[i] = new BranchClass();            // create new BranchClass
            branch[i] = tmp.branch[i];
        }
    }

/**
 *  Initialize one branch cell in a node.
 */
    public void    RTreeInitBranch( BranchClass b ) {
        b.rect.RTreeInitRect();
        b.child = null;
    }

/**
 *  Initialize a NodeClass
 */
    public void    RTreeInitNode( ) {
        count = 0;
        level = -1;
        for( int i = 0; i < NODECARD; i++ )
            branch[i] = new BranchClass();
    }

/**
 *  Method function for Display the spatial database
 */
    public void    RTreePrintAll( ) {
        StringBuffer    s = new StringBuffer();

        RTreePrintNode(0, s);
        new DisplayClass( s );
    }

/**
 *  Method function for Print branch
 */
    public void    RTreePrintBranch(BranchClass b, int depth, StringBuffer s) {
        b.rect.RTreePrintRect(depth, s);
        b.child.RTreePrintNode(depth, s);
    }
```

```
/**
 *  Method function for Print Node
 */
    public  void    RTreePrintNode(int depth, StringBuffer s) {
        branch[0].rect.RTreeTabIn(depth, s);
        if( level == 0 )
            s.append(" LEAF");
        else    if( level > 0 )
                    s.append(" NONLEAF");
                else
                    s.append(" TYPE=?");
        s.append("  level = " + level + "  count = " + count + "\n");
        for( int i = 0; i < count; i++ ) {
            if( level == 0 ) {      // Leaf node
                branch[0].rect.RTreeTabIn(depth, s);
                s.append("\t" + i + ":  record = " + branch[i].rect.record + "\n");
            }
            else {      // Non-leaf node
                branch[0].rect.RTreeTabIn(depth, s);
                s.append("branch " + i + "\n");
                RTreePrintBranch(branch[i], depth+1, s);
            }
        }
    }

/**
 *  Method function for save spatial database to local file.
 */
    public  void    RTreeSaveData( RandomAccessFile file ) {
        byte    temp[];
        int     size;

        try {
            for( int i = 0; i < count; i++ )
                if( level == 0 ) {
                    for( int j = 0; j < branch[i].rect.bound.length; j++ )
                    file.writeFloat( branch[i].rect.bound[j] );
                    size = branch[i].rect.record.length();
                    temp = new byte[size];
                    branch[i].rect.record.getBytes(0, size, temp, 0);
                    file.writeInt( size );
                    file.write( temp );
                }
                else
                branch[i].child.RTreeSaveData( file );
        } catch( IOException e ) {
            System.err.println( e );
        }
    }

/**
 *  Find the smallest rectangle that includes all rectangles in rect of a node
 */
    public RectClass RTreeNodeCover( ) {
        RectClass r = new RectClass();

        if( !branch[0].rect.RTreeIsEmpty() )
            r = branch[0].rect;
        for( int i = 1; i < NODECARD; i++ ) {
            if( !branch[i].rect.RTreeIsEmpty() )
            r = r.RTreeCombine( branch[i].rect );
        }
        return r;
    }

/**
 *  Pick a branch.  Pick the one that will need the smallest increase in area
 *  to accommodate the new rectangle.  This will result in the least total area
 *  for the convering rectangles in the current node.  In case of a tie, pick
 *  the one which was smaller before, to get the best resolution when
 *  searching.
 */
    public int RTreePickBranch( RectClass R ) {
        RectClass  r = new RectClass();
        RectClass  temp = new RectClass();
        double     area, increase;
        double     bestIncr = (double)-1.0;
        double     bestArea = (double)0.0;
        boolean    first_time = true;
        int        best = 0;

        for( int i = 0; i < NODECARD; i++ )
            if( branch[i].child != null ) {
                r = branch[i].rect;
                area = r.RTreeVolume();
                temp = R.RTreeCombine( r );
                increase = temp.RTreeVolume() - area;
                if( increase < bestIncr  ||  first_time ) {
                    best = i;
                    bestArea = area;
                    bestIncr = increase;
                    first_time = false;
                } else
                    if( increase == bestIncr  &&  area < bestArea ) {
                        best = i;
                        bestArea = area;
                        bestIncr = increase;
                    }
            }
        return  best;
    }

/**
 *  Add a branch to a node.  Split the node if necessary.
 *  Returns false if node not split.  Old node update.
 *  Returns true if node split, sets new_node to address of new node.
 *  Old node update, becomes one of two.
 */
    public  boolean     RTreeAddBranch( BranchClass b, NodeClass N ) {
        if( count < NODECARD ) {                        // split won't be necessary
            for(int i = 0; i < NODECARD; i++ )         // find empty branch
                if( branch[i].rect.RTreeIsEmpty() ) {
                    branch[i] = b;
                    count++;
                    break;
                }
            return  false;          // Not Split
        } else {
            RTreeSplitNode( b, N );
            return  true;           // Split
        }
    }

/**
 *  Disconnect a dependent node.
 */
    public  void    RTreeDisconnectBranch( int i ) {
        count--;
        for( int j = i; j < count; j++ )
            branch[j] = branch[j+1];
        branch[count] = new BranchClass();
    }

/**
```

```
* Split a node.
* Divides the nodes branches and the extra one between two nodes.
* Old node is one of the new ones, and one really new one is created.
* Tries more than one method for choosing a partition, uses best result
*/
    public void    RTreeSplitNode( BranchClass b, NodeClass nn ) {
        SplitClass p = new SplitClass();
        int          tmp_level;

    /**
    *   load all the branches into a buffer, initialize old node
    */
        tmp_level = level;
        p.RTreeGetBranches( this, b );

    /**
    *   find partition
    */
        p.RTreeMethodZero();

    /**
    *   put branches from buffer into 2 nodes according to chosen partition
    *   Will create new space different from original
    */
        nn.level = level = tmp_level;
        p.RTreeLoadNodes( this, nn );
    }
};
```

```
/*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
//
//      AUTHOR                  : VEERA ASVASERMCHAROEN
//      COURSE                  : COMSC 5000 - Thesis
//      TOPIC                   : Multi-dimensional spatial database system.
//      PROGRAM NAME            : SplitClass.java
//      PACKAGE FILE            : RTree
//      IMPORT FILE             : java.lang.*
//      MACHINE                 : SPARC memory 64 Mb
//      OPERATING SYSTEM        : Solaris 2.4
//      LANGUAGE                : java language
//      COMPILER                : java compiler from Sun Microsystems version 1.0.2
//      DATE                    : July, 1996
//
/*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */

package    RTree;

import java.lang.*;

public class    SplitClass extends ConstClass {
    /**
     *   static variables mean that these variables have only one copy for all classes
     */
    static private   BranchClass    BranchBuf[] = new BranchClass[NODECARD+1];
    static private   RectClass      CoverSplit = new RectClass();
    static private   float          CoverSplitArea;

    int          partition[] = new int[NODECARD+1];
    boolean      taken[] = new boolean[NODECARD+1];
    RectClass  cover[] = new RectClass[2];
    float        area[] = new float[2];          // Rect Type
    int          count[] = new int[2];

/**
 *   SplitClass Constructor
 */
    SplitClass( ) {
        for( int i = 0; i <= NODECARD; i++ )
            BranchBuf[i] = new BranchClass();
        cover[0] = new RectClass();
        cover[1] = new RectClass();
        count[0] = count[1] = 0;
    }
```

```
/**
 *  Load branch buffer with branches from full node plus the extra branch
 */
   public  void    RTreeGetBranches( NodeClass n, BranchClass b ) {
       for( int i = 0; i < NODECARD; i++ )          // load the branch buffer
           BranchBuf[i] = n.branch[i];
       BranchBuf[NODECARD} = b;
       /**
       *   calculate rect containing all int the set
       */
       CoverSplit = BranchBuf[0].rect;
       for( int i = 0; i <= NODECARD; i++ )
           CoverSplit = CoverSplit.RTreeCombine( BranchBuf[i].rect );
       CoverSplitArea = CoverSplit.RTreeVolume();
       n.RTreeInitNode();
   }

/**
 *  Put a branch in one of the groups.
 */
   public  void    RTreeClassify( int i, int group ) {
       partition[i] = group;
       taken[i] = true;
       if( count[group] == 0 )
           cover[group] = BranchBuf[i].rect;
       else
           cover[group] = BranchBuf[i].rect.RTreeCombine( cover[group] );
       area[group] = cover[group].RTreeVolume();
       count[group]++;
   }

/**
 *  Pick two rects from set to be the first elements of the two groups.
 *  Pick the two that waste the most area if covered by a single rectangle.
 */
   public  void    RTreePickSeeds( ) {
       RectClass  one_rect;
       int           seed0 = 0, seed1 = 0;
       float        tmp_area[] = new float[NODECARD + 1];    // Rect Type
       float        worst;      // Rect Type
       float        waste;      // Rect Type

       for( int i = 0; i < NODECARD; i++ )
           tmp_area[i] = BranchBuf[i].rect.RTreeVolume();
       worst = -CoverSplitArea - 1;
```

```
        for( int i = 0; i < NODECARD; i++ )
            for( int j = i+1; j <= NODECARD; j++ ) {
                one_rect = BranchBuf[i].rect.RTreeCombine( BranchBuf[j].rect );
                waste = one_rect.RTreeVolume() - tmp_area[i] - tmp_area[j];
                if( waste > worst ) {
                    worst = waste;
                    seed0 = i;
                    seed1 = j;
                }
            }
        RTreeClassify( seed0, 0 );
        RTreeClassify( seed1, 1 );
    }

/**
 *  Copy branches from the buffer into two nodes according to the partition.
 */
    public  void    RTreeLoadNodes( NodeClass n, NodeClass q ) {
        for( int i = 0; i <= NODECARD; i++ )
            switch( partition[i] ) {
                case    0:
                    n.RTreeAddBranch( BranchBuf[i], null );
                    break;
                case    1:
                    q.RTreeAddBranch( BranchBuf[i], null );
                    break;
            }
    }

/**
 *  Initialize a SplitClass
 */
    public  void    RTreeInitSplit( ) {
        count[0] = count[1] = 0;
        cover[0].RTreeNull();
        cover[1].RTreeNull();
        area[0] = area[1] = (float)0.0;
        for( int i = 0; i <= NODECARD; i++ ) {
            taken[i] = false;
            partition[i] = -1;
        }
    }

/**
 *  Print out data for partition
```

```
*/
    public void    RTreePrintSplit( ) {
        StringBuffer    s = new StringBuffer();

        s.append("\n" + "partition");
        for( int i = 0; i <= NODECARD; i++ )
            s.append( i );
        s.append(" ");
        for( int i = 0; i <= NODECARD; i++ )
            if( taken[i] )
                s.append("   t\t");
            else
                s.append("\t");
        s.append(" " + "\n");
        for( int i = 0; i <= NODECARD; i++ )
            s.append( partition[i] );
        s.append(" " + "\n");
        s.append("count[0] = " + count[0] + " area = " + area[0] + "\n");
        s.append("count[1] = " + count[1] + " area = " + area[1] + "\n");
        if( area[0] + area[1] > 0 )
            s.append("total area = " + area[0] + area[1] + "effectiveness = " +
                            (float)CoverSplitArea / (area[0] + area[1]) + "\n");
        s.append("cover[0]:" + "\n");
        cover[0].RTreePrintRect(0, s);
        s.append("cover[1]:" + "\n");
        cover[1].RTreePrintRect(0, s);
        new DisplayClass( s );
    }

/**
 *  Method #0 for choosing a partition:
 *  As the seeds for the two groups, pick the two rects that would waste the
 *  most area if covered by a single rectangle, i.e. evidently the worst pair to have in the
 *  same group.  Of the remaining, one at a time is chosen to be put in one of the two
 *  groups.  The one chosen is the one with the greatest difference in area
 *  expansion depending on which group - the rect most strongly attracted to
 *  one group and repelled from the other.
 *  If one group gets too full (more would force other group to violate min fill
 *  requirement) then other group gets the rest.  These last are the ones that can go in
 *  either group most easily.
 */
    public void    RTreeMethodZero( ) {
        float    biggestDiff;
        int      group, chosen = 0, betterGroup = 0;
        RTreeInitSplit();
        RTreePickSeeds();
        while( ( (count[0] + count[1]) <= NODECARD )
                            && ( count[0] < (NODECARD - MinFill + 1) )
                            && ( count[1] < (NODECARD - MinFill + 1) ) ) {
            biggestDiff = Float.MIN_VALUE;
            for( int i = 0; i <= NODECARD; i++ ) {
                if( !taken[i] ) {
                    RectClass   r, rect_0, rect_1;
                    float    growth0, growth1, diff;

                    r = BranchBuf[i].rect;
                    rect_0 = r.RTreeCombine( cover[0] );
                    rect_1 = r.RTreeCombine( cover[1] );
                    growth0 = rect_0.RTreeVolume() - area[0];
                    growth1 = rect_1.RTreeVolume() - area[1];
                    diff = growth1 - growth0;
                    if( diff >= 0 )         group = 0;
                    else {
                        group = 1;
                        diff = -diff;
                    }
                    if( diff > biggestDiff ) {
                        biggestDiff = diff;
                        chosen = i;
                        betterGroup = group;
                    }
                    else
                        if( diff==biggestDiff && count[group]<count[betterGroup] ) {
                            chosen = i;
                            betterGroup = group;
                        }
                }
            }
            RTreeClassify( chosen, betterGroup );
        }
        //  if one group too full, put remaining rects in the other
        if( count[0] + count[1] <= NODECARD ) {
            if( count[0] >= NODECARD + 1 - MinFill )        group = 1;
            else            group = 0;
            for( int i = 0; i <= NODECARD; i++ )
                if( !taken[i] )         RTreeClassify( i, group );
        }
    }
};
```

```
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
//
//      AUTHOR              : VEERA ASVASERMCHAROEN
//      COURSE              : COMSC 5000 - Thesis
//      TOPIC               : Multi-dimensional spatial database system.
//      PROGRAM NAME        : IndexClass.java
//      PACKAGE FILE        : RTree
//      IMPORT FILE         : java.lang.*
//      MACHINE             : SPARC memory 64 Mb
//      OPERATING SYSTEM    : Solaris 2.4
//      LANGUAGE            : java language
//      COMPILER            : java compiler from Sun Microsystems version 1.0.2
//      DATE                : July, 1996
//
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */

package   RTree;

import java.lang.*;

public class   IndexClass    extends ConstClass {
    IndexClass     next;
    NodeClass      node;

/**
 *  Constructor IndexClass
 */
    public IndexClass( ) {
        next = null;
        node = new NodeClass();
    }

/**
 *  Overriding Constructor IndexClass
 */
    public IndexClass( int dim ) {
        setNumDims( dim );
        next = null;
        node = new NodeClass();
    }

/**
 *  Make a new index, empty.  Consists of a single node.
 */
    public NodeClass     RTreeNewIndex( ) {
        NodeClass  x = new NodeClass();             // for init level set to -1

        x.level = 0;        // leaf node
        return x;
    }

/**
 *  Search in an index tree or subtree for all data rectangle that
 *  overlap the argument rectangle, and set the data objects to the object
 *  that covered by the given rectangle
 *  Returns the number of qualifying data rects.
 */
    public  int  RTreeSearch( NodeClass n, RectClass r ) {
        int     hitCount = 0;

        if( n.level > 0 ) {     // this is an internal node in the tree
            for( int i = 0; i < NODECARD; i++ )
                if( n.branch[i].child != null  &&  r.RTreeOverlap( n.branch[i].rect ) )
                    hitCount += RTreeSearch( n.branch[i].child, r );
        } else                  // this is a leaf node
            for( int i = 0; i < NODECARD; i++ )
                if( !n.branch[i].rect.RTreeIsEmpty() &&
                                                r.RTreeContained( n.branch[i].rect ) ) {
                    hitCount++;
                    r.record = n.branch[i].rect.record;
                }
        return  hitCount;
    }

/**
 *  Inserts a new data rectangle into the spatial data structure.
 *  Recursively descends tree, propagates splits back up.
 *  Returns false if node was not split.  Old node updated.
 *  If node was split, returns true and sets the pointer
 *  point to the new node.  Old node updated to become one of two.
 *  The level argument specifies the number of steps up from the leaf
 *  level to insert; e.g. a data rectangle goes in at level = 0
 */
    public  boolean     RTreeInsertRect2( BranchClass br, NodeClass n,
                                            NodeClass new_node, int level ) {
        int             i;
        BranchClass     b = new BranchClass( );
        NodeClass       n2 = new NodeClass( );

        /**
```

```
         *  Still above level for insertion, go down tree recursively
         */
        if( n.level > level ) {
            i = n.RTreePickBranch( br.rect );
            if( !RTreeInsertRect2(br, n.branch[i].child, n2, level ) ) {
                /**
                 *  child was not split
                 */
                n.branch[i].rect = br.rect.RTreeCombine( n.branch[i].rect );
                return  false;
            } else {            // child was split
                n.branch[i].rect = n.branch[i].child.RTreeNodeCover();
                b.child = n2;
                b.rect = n2.RTreeNodeCover();
                return  n.RTreeAddBranch( b, new_node );
            }
        }
        else if( n.level == level ) {
                b = br;
                /**
                 * child field of leaves contains record of data record
                 */
                return  n.RTreeAddBranch( b, new_node );
            }
        return  false;
    }

/**
 *  Insert a data rectangle into the spatial data structure.
 *  RTreeInsertRect provides for splitting the root;
 *  return root if root was not split, if root was split, return newroot
 *  The level argument specifies the number of steps up from the leaf
 *  level to insert; e.g. a data rectangle goes in at level = 0.
 *  RTreeInsertRect2 does the recursion
 */
    public  NodeClass RTreeInsertRect(BranchClass branch, NodeClass root, int level) {
        NodeClass       newnode = new NodeClass();

        if( RTreeInsertRect2(branch, root, newnode, level) ) {          // root split
            NodeClass       newroot = new NodeClass();
            BranchClass    b = new BranchClass();

            /**
             *  grow a new root & tree taller
             */
            newroot.level = root.level + 1;
            b.rect = root.RTreeNodeCover();
            b.child = root;
            newroot.RTreeAddBranch( b, null );
            b = new BranchClass();          // create new branch
            b.rect = newnode.RTreeNodeCover();
            b.child = newnode;
            newroot.RTreeAddBranch( b, null );
            return  newroot;
        } else
            return  root;          // root does not split
    }

/**
 *  Overloading Method RTreeInsertRect
 */
    public  NodeClass      RTreeInsertRect( RectClass r, NodeClass root, int level ) {
        BranchClass    br = new BranchClass();

        br.rect = r;
        return  RTreeInsertRect(br, root, level);
    }

/**
 *  Add a node to the reinsertion list.  All its branches will later
 *  be reinserted into the index structure.
 */
    public  IndexClass      RTreeReInsert( NodeClass n, RootClass ee ) {
        IndexClass  l = new IndexClass();

        /**
         *  create link list
         */
        l.node = n;
        l.next = ee.root;
        return l;
    }

/**
 *  Delete a rectangle from non-root part of an index structure.
 *  Called by RTreeDeleteRect.  Descends tree recursively,
 *  merges branches on the way back up.
 */
    public  boolean     RTreeDeleteRect2( RectClass r, NodeClass n, RootClass ee ) {
        if( n.level > 0 ) {                    // not a leaf node
```

```
            for( int i = 0; i < NODECARD; i++ )
                if( n.branch[i].child != null  &&  r.RTreeOverlap( n.branch[i].rect ) ) {
                    if( !RTreeDeleteRect2(r, n.branch[i].child, ee) ) {
                        if( n.branch[i].child.count >= MinFill )
                            n.branch[i].rect = n.branch[i].child.RTreeNodeCover();
                        else {      // not enough entries in child, eliminate child node
                            ee.root = RTreeReInsert( n.branch[i].child, ee );
                            n.RTreeDisconnectBranch( i );
                        }
                        return  false;
                    }
                }
            return  true;
        }
        else {      // a leaf node
            for( int i = 0; i < NODECARD; i++ ) {
                if( r.RTreeContained( n.branch[i].rect ) &&
                                        n.branch[i].rect.record.equals( r.record ) ) {
                    n.RTreeDisconnectBranch( i );
                    return  false;
                }
            }
            return  true;
        }
    }

/**
 *  Delete a data rectangle from an index structure.
 *  Pass in a pointer to a Rect, ptr to ptr to root node.
 *  Returns NodeClass nn.
 *  RTreeDeleteRect provides for eliminating the root.
 */
    public  NodeClass     RTreeDeleteRect( RectClass r, NodeClass nn ) {
        NodeClass     tmp_node = new NodeClass();
        RootClass     reInsertList = new RootClass();
        IndexClass    e = new IndexClass();

        if( !RTreeDeleteRect2(r, nn, reInsertList) ) {
            /**
             *  found and delete a data item reinsert and branches from eliminated nodes
             */
            while( reInsertList.root != null ) {
                tmp_node = reInsertList.root.node;
                for( int i = 0; i < NODECARD; i++ )
                    if( !tmp_node.branch[i].rect.RTreeIsEmpty() )
                    nn=RTreeInsertRect(tmp_node.branch[i], nn, tmp_node.level);
                reInsertList.root = reInsertList.root.next;
            }
            /**
             *  check for redundant root (not leaf, 1 child) and eliminate
             */
            if( nn.count == 1  &&  nn.level > 0 ) {
                for( int i = 0; i < NODECARD; i++ ) {
                    tmp_node = nn.branch[i].child;
                    if( tmp_node != null )
                        break;
                }
                nn = tmp_node;
            }
            return  nn;
        }
        return  nn;
    }
};
```

```
/* * * * * * * * * * * * * * * * * * * * * * * */
//
//      AUTHOR                : VEERA ASVASERMCHAROEN
//      COURSE                : COMSC 5000 - Thesis
//      TOPIC                 : Multi-dimensional spatial database system.
//      PROGRAM NAME          : DisplayClass.java
//      PACKAGE FILE          : RTree
//      IMPORT FILE           : java.awt.*
//      MACHINE               : SPARC memory 64 Mb
//      OPERATING SYSTEM      : Solaris 2.4
//      LANGUAGE              : java language
//      COMPILER              : java compiler from Sun Microsystems version 1.0.2
//      DATE                  : July, 1996
//
/* * * * * * * * * * * * * * * * * * * * * * * */

package   RTree;

import java.awt.*;

public class   DisplayClass  extends Frame {
    /**
     *  "final" variables are constant
     *  "static" variables mean there are only one copy for all classes
     */
    private static final int        H_SIZE = 550;
    private static final int        V_SIZE = 350;
    private static final String  DISMISS = " Dismiss";

    public DisplayClass( StringBuffer buffer ) {
        super(" Display Area ");
        TextArea   textArea = new TextArea(90, 65);
        Button       b = new Button( DISMISS );

        textArea.setFont( new Font("TimesRoman", Font.PLAIN, 18) );
        b.setFont( new Font("TimesRoman", Font.ITALIC, 19) );
        setBackground( Color.yellow );
        setForeground( Color.black );
        b.setBackground( Color.black );
        b.setForeground( Color.green );
        textArea.setText( buffer.toString() );
        textArea.setEditable( false );
        add("Center", textArea);
        add("South", b);
        pack();
        resize(H_SIZE, V_SIZE);
        show();
    }

    public boolean    handleEvent( Event evt ) {
        switch( evt.id ) {
            case    Event.WINDOW_DESTROY:
                dispose();
                return( true );
            case    Event.ACTION_EVENT:
                if( DISMISS.equals( evt.arg ) ) {
                    dispose();
                    return( true );
                }
        }
        return( false );
    }
};
```

```
/*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
//
//      AUTHOR                     : VEERA ASVASERMCHAROEN
//      COURSE                     : COMSC 5000 - Thesis
//      TOPIC                      : Multi-dimensional spatial database system.
//      PROGRAM NAME               : mainClass.java
//      IMPORT USER FILE           : RTree.*
//      IMPORT SYSTEM FILE         : java.io.*  java.awt.* java.net.*  java.lang.*
//      PACKAGE FILE               : RTree
//      IMPORT FILE                : java.awt.*
//      MACHINE                    : SPARC memory 64 Mb
//      OPERATING SYSTEM           : Solaris 2.4
//      LANGUAGE                   : java language
//      COMPILER                   : java compiler from Sun Microsystems version 1.0.2
//      DATE                       : July, 1996
//
/*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
public class   mainClass extends Frame
{
    IndexClass      list = null;
    RectClass       r = null;
    NodeClass       root = null;        // root of the spatial data structure
    float           rectField[];            // rectangle field
    String          recordField;        // object record variable
    int             dimension;          // number of dimension
    int             hit;
    boolean         flag;
    String          FileName;

    /**
     *  "final" variables are constants
     */
    final static String  TIMES_ROMAN = " Times Roman ",
                         HELVETICA   = " Helvetica ",
                         SYSTEM      = " System ";
    final static String  DISMISS     = " Dismiss ",
                         FILE        = " File ",
                         NEW         = " New ",
                         OPEN        = " Open ",
                         SAVE        = " Save ",
                         URL_READ    = " Read from URL ",
                         DISPLAY     = " Display ",
                         OK          = " OK ",
                         CANCEL      = " Cancel ",
                         UPDATE      = " Update ",
                    SEARCH          = " Search ",
                    INSERT          = " Insert ",
                    DELETE          = " Delete ",
                    STATUS          = " Show Status ",
                    QUIT            = " Quit ",
                    EXIT            = " Exit ",
                    HELP            = " Help ",
                    REPORT          = " Report Bug ",
                    ABOUT           = " About ";
    final static int H_SIZE         = 700,
                    V_SIZE          = 300;

    StatusBar       sb;        // Object variable for show status line
    ToolBar         tb;        // Object variable for show menu

    private FileDialog              openDialog, saveDialog;
    private HelpDialog              helpDialog = null;
    private ReportDialog            reportDialog = null;
    private AboutDialog             aboutDialog = null;
    private NewDialog               newDialog = null;
    private ErrorDialog             errorDialog = null;
    private DisplayDialog           displayDialog = null;
    private DisplaySearchDialog     displaySearchDialog = null;
    private ReadURLDialog           readURLDialog = null;

    private CheckboxMenuItem        buttonMenuItem;

/**
 *  Constructor mainClass for initialized variable and set up the screen
 */
    public mainClass( ) {
        /**
         * Calls the parent constructors Frame( String Title )
         * Equivalent to setTitle(" Multi-dimensional Spatial DataBase ");
         */
        super(" Multi-dimensional Spatial DataBase ");

        CreateDialog();         // create open and save dialog
        InitializeMenus();      // set menu bar

        setBackground( Color.lightGray );
        setForeground( Color.black );

        add("North", tb = new ToolBar() );
        add("South", sb = new StatusBar() );
    pack();
    resize(H_SIZE, V_SIZE);
    show();
}

public  boolean    handleEvent( Event evt ) {
    /**
     *  Return false is we want the system to also process the message,
    *   otherwise return true to say we're done with the message
     */
    switch( evt.id ) {
        /**
         *  Event.WINDOW_DESTROY documentation can be found
         *  in the Event classes
         */
        case    Event.WINDOW_DESTROY:
            stop();      // want to quit the application
        case    Event.ACTION_EVENT:
        {
            showStatus( evt.arg.toString() );
            if( HELP.equals( evt.arg ) )
                showHelpDialog();
            else  if( REPORT.equals( evt.arg ) )
                    showReportDialog();
            else  if( ABOUT.equals( evt.arg ) )
                    showAboutDialog();
            else  if( NEW.equals( evt.arg ) )
                    showNewDialog();
            else  if( OPEN.equals( evt.arg ) )
                    openFileDialog();
            else  if( SAVE.equals( evt.arg ) )
                    saveFileDialog();
            else  if( URL_READ.equals( evt.arg ) )
                    showReadURL();
            else  if( DISPLAY.equals( evt.arg ) )
                    displayDialog();
            else  if( evt.target instanceof Choice )
                    showDisplayDialog( evt.arg.toString() );
            else  if( SEARCH.equals( evt.arg ) )
                    showDisplayDialog( SEARCH );
            else  if( INSERT.equals( evt.arg ) )
                    showDisplayDialog( INSERT );
            else  if( DELETE.equals( evt.arg ) )
                    showDisplayDialog( DELETE );
                else  if( QUIT.equals( evt.arg ) || EXIT.equals( evt.arg ) )
                        stop();
                showStatus( evt.arg.toString() );
                return true;
            }
            default:
                return false;
        }
    }

/**
 *  Method function to create open and save dialog
 */
    public  void    CreateDialog( ) {
        openDialog = new FileDialog(this, " Open Spatial DataBase File ",
                                                FileDialog.LOAD);
        saveDialog = new FileDialog(this, " Save Spatial DataBase File ",
                                                FileDialog.SAVE);
    }

/**
 *  Method function for open local spatial database
 */
    public  void    openFileDialog( ) {
        String  openFileName;

        openDialog.show();
        openFileName = openDialog.getFile();
        if( openFileName != null ) {
            FileName = check( openFileName );
            try {
                RandomAccessFile     file = new RandomAccessFile(FileName, "r");

                System.out.println("Read data to file \"" + FileName + "\"");
                dimension = file.readInt();
                list = new IndexClass( dimension );
                root = list.RTreeNewIndex();

                while( file.getFilePointer() < file.length() ) {
                    ReadRect( file );
                    r = new RectClass( rectField, recordField );
                    root = list.RTreeInsertRect( r, root, 0 );
                }
                file.close();
            } catch( IOException e ) {
                showErrorDialog("Can't read \"" + FileName + "\"");
                System.err.println( e );
            }
        }
    }

/**
 *  Method function for save local spatial database
 */
    public void    saveFileDialog( ) {
        String  saveFileName;

        saveDialog.show();
        saveFileName = saveDialog.getFile();
        if( saveFileName != null ) {
            FileName = check( saveFileName );
            try {
                RandomAccessFile  file = new RandomAccessFile(FileName, "rw");

                System.out.println("Write data to file \"" + FileName + "\"");
                file.writeInt( dimension );
                root.RTreeSaveData( file );
                file.close();
            } catch( IOException e ) {
                showErrorDialog("Can't write \"" + FileName + "\"");
                System.err.println( e );
            }
        }
    }

/**
 *  Compensate for dialog bug
 */
    public String  check( String filename ) {
        if( filename.endsWith(".*.*") ) {
            filename = filename.substring(0, filename.length()-4);
        }
        return( filename );
    }

/**
 *  Method function for initialize and show menu bar
 */
    public void    InitializeMenus() {
        MenuBar        mbar = new MenuBar();
        Menu          m;
        MenuItem      mi;

        m = new Menu( FILE );
        m.add( new MenuItem( NEW ) );
        m.add( new MenuItem( OPEN ) );
        m.add( new MenuItem( SAVE ) );
        m.addSeparator();
        m.add( new MenuItem( URL_READ ) );
        m.addSeparator();
        buttonMenuItem = new CheckboxMenuItem( STATUS );
        buttonMenuItem.setState( true );
        m.add( buttonMenuItem );
        m.addSeparator();
        m.add( new MenuItem( QUIT ) );
        mbar.add( m );

        m = new Menu( UPDATE );
        m.add( new MenuItem( SEARCH ) );
        m.add( new MenuItem( INSERT ) );
        m.add( new MenuItem( DELETE ) );
        m.add( new MenuItem( DISPLAY ) );
        mbar.add( m );

        m = new Menu( HELP );
        m.add( new MenuItem( HELP ) );
        m.addSeparator();
        m.add( new MenuItem( REPORT ) );
        m.addSeparator();
        m.add( new MenuItem( ABOUT ) );
        mbar.add( m );

        // Recent change
        setMenuBar( mbar );
    }

/**
 *  Method function for calling Object StatusBar
 */
    public  void    showStatus( String s ) {
        if( buttonMenuItem.getState() )
            sb.showStatus(s, dimension, FileName);
        else
            sb.hideStatus();
    }
```

```
/**
 *  Method function for reading remote spatial database file
 */
    public void    ReadRect( RandomAccessFile file ) {
        byte    temp[];
        int     size;

        rectField = new float[dimension * 2];
        try {
            for( int i = 0; i < rectField.length; i++ )
                rectField[i] = file.readFloat();
            size = file.readInt();
            temp = new byte[size];
            file.read( temp );
            recordField = new String(temp, size);
        } catch( IOException e ) {
            showErrorDialog("SomeThing was Wrong #2");
            System.err.println( e );
        }
    }

/**
 *  Method function for write local spatial database
 */
    public void    WriteRect( RandomAccessFile file ) {
        byte    temp[];
        int     size;

        try {
            for( int i = 0; i < rectField.length; i++ )
                file.writeFloat( rectField[i] );
            size = recordField.length();
            temp = new byte[size];
            recordField.getBytes(0, size, temp, 0);
            file.writeInt( size );
            file.write( temp );
        } catch( IOException e ) {
            showErrorDialog("SomeThing was Wrong #3");
            System.err.println( e );
        }
    }

/**
 *  Method function show help window
 */
    public void    showHelpDialog( ) {
        if( helpDialog != null )
            helpDialog.dispose();
        helpDialog = new HelpDialog( this );
        helpDialog.show();
    }

/**
 *  Method function show report window
 */
    public void    showReportDialog( ) {
        if( reportDialog != null )
            reportDialog.dispose();
        reportDialog = new ReportDialog( this );
        reportDialog.show();
    }

/**
 *  Method function show about window
 */
    public void    showAboutDialog( ) {
        if( aboutDialog != null )
            aboutDialog.dispose();
        aboutDialog = new AboutDialog( this );
        aboutDialog.show();
    }

/**
 *  Method function show new window for new spatial data structure
 */
    public void    showNewDialog( ) {
        if( newDialog != null )
            newDialog.dispose();
        newDialog = new NewDialog( this );
        newDialog.show();
        if( flag )
            root = null;
            flag = false;
    }

/**
 *  Method function show read the spatial database from URL window
 */
    public void    showReadURL( ) {
        if( readURLDialog != null )
            readURLDialog.dispose();
        readURLDialog = new ReadURLDialog( this );
        readURLDialog.show();
        if( flag ) {
            try {
                new ReadFileURL(this, new URL( FileName )):
            }
            catch( MalformedURLException e ) {
                showErrorDialog( e.toString() );
            }
            catch( IOException e ) {
                showErrorDialog( e.toString() );
            }
        }
        flag = false;
    }

/**
 *  Method function show output window
 */
    public  void    displayDialog( ) {
        if( root != null )
            root.RTreePrintAll();
        else
            showErrorDialog("No Data in this Structure to Display");
    }

/**
 *  Method function show display window
 */
    public  synchronized  void    showDisplayDialog( String showString ) {
        if( dimension <= 0 ) {
            showErrorDialog("ERROR IN DIMENSION NUMBER");
            return;
        }
        if( displayDialog != null )
            displayDialog.dispose();
        displayDialog = new DisplayDialog(this, showString);
        displayDialog.show();
        if( flag ) {
            flag = false;
            r = new RectClass(rectField, recordField);
            if( showString.equals( SEARCH ) )
                if( root == null )
```

```
                    showErrorDialog("This Spatial Data Structure is Empty");
                else {
                    hit = 0;
                    hit = list.RTreeSearch(root, r);
                    if( hit != 0 ) {
                        if( displaySearchDialog != null )
                            displaySearchDialog.dispose();
                            recordField = r.RTreeGetRecord();
                            displaySearchDialog = new DisplaySearchDialog( this );
                            displaySearchDialog.show();
                        } else
                            showErrorDialog("HIT NUMBER = 0,  Record not Found");
                    }
                    else  if( showString.equals( INSERT ) ) {
                            if( root == null )
                                root = list.RTreeNewIndex();
                            root = list.RTreeInsertRect(r, root, 0);
                        }
                        else  if( showString.equals( DELETE ) )
                            if( root == null )
                                showErrorDialog("This Spatial Data Structure is Empty");
                            else
                                root = list.RTreeDeleteRect(r, root);
        }
    }

/**
 *  Method function show error window
 */
    public  void    showErrorDialog( String err ) {
        if( errorDialog != null )
            errorDialog.dispose();
        errorDialog = new ErrorDialog(this, err);
        errorDialog.show();
    }

    public  void    stop( ) {
        System.exit( 0 );
    }

    /**
     *  all variables in this function are static.
     */
    public  static   void    main( String args[] ) {
        new mainClass();
    }
};


/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
 *
 *  Class  StatusBar
 *
 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
class   StatusBar  extends Panel
{
    Label       infoLabel;
    Label       fileNameLabel;
    Label       dimensionLabel;

/**
 *  Constructor for initialize all variable
 */
    public  StatusBar( ) {
        setLayout( new FlowLayout() );
        infoLabel = new Label(" StatusBar Created ");
        infoLabel.setFont( new Font("TimesRoman", Font.BOLD, 20) );
        add( infoLabel );
        dimensionLabel = new Label(" Dimension : 0  ");
        dimensionLabel.setFont( new Font("TimesRoman", Font.BOLD, 20) );
        add( dimensionLabel );
        fileNameLabel = new Label(" No File                    ");
        fileNameLabel.setFont( new Font("TimesRoman", Font.ITALIC, 20) );
        add( fileNameLabel );
    }

/**
 *  Method function for hiding status line
 */
    public  void    hideStatus( ) {
        infoLabel.setText(" ");
        dimensionLabel.setText(" ");
        fileNameLabel.setText(" ");
    }

/**
 *  Method function for showing a new status line
 */
    public  void    showStatus(String status, int dim, String fileName) {
        infoLabel.setText( status );
        dimensionLabel.setText(" Dimension : " + dim);
        if( fileName == null )
            fileNameLabel.setText(" No File                    ");
        else
            fileNameLabel.setText( fileName );
    }
};


/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
 *
 *  Class  ToolBar
 *
 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
class   ToolBar   extends Panel
{
/**
 *  Constructor for showing menu bar
 */
    public  ToolBar( ) {
        Button  b;

        setLayout( new FlowLayout() );
        b = new Button( mainClass.NEW );
        b.setForeground( Color.white );
        b.setBackground( Color.black );
        b.setFont( new Font("Helvetica", Font.ITALIC, 19) );
        add( b );

        b = new Button( mainClass.URL_READ );
        b.setForeground( Color.white );
        b.setBackground( Color.black );
        b.setFont( new Font("Helvetica", Font.ITALIC, 19) );
        add( b );

        b = new Button( mainClass.DISPLAY );
        b.setForeground( Color.white );
        b.setBackground( Color.black );
        b.setFont( new Font("Helvetica", Font.ITALIC, 19) );
        add( b );

        Choice c = new Choice();
        c.setForeground( Color.pink );
        c.setBackground( Color.darkGray );
        c.setFont( new Font("Helvetica", Font.ITALIC, 19) );
        c.addItem( mainClass.SEARCH );
        c.addItem( mainClass.INSERT );
        c.addItem( mainClass.DELETE );
        add( c );

        b = new Button( mainClass.EXIT );
        b.setForeground( Color.green );
        b.setBackground( Color.black );
        b.setFont( new Font("Helvetica", Font.ITALIC, 19) );
        add( b );
    }
};


/* * * * * * * * * * * * * * * * * * * * * * * */
 *
 *  Class  HelpDialog
 *
 * * * * * * * * * * * * * * * * * * * * * * * */
class   HelpDialog    extends Dialog {
    mainClass       parent;

    public  HelpDialog( mainClass parent ) {
        super(parent, " Help Window ", true);
        setForeground( Color.blue );
        setBackground( Color.lightGray );
        this.parent = parent;

        int             WIDTH = 500;
        int             HEIGHT = 200;
        Dimension       d;

        d = parent.size();
        setButton();
        resize(WIDTH, HEIGHT);
        setResizable( false );
    }

    public  boolean     handleEvent( Event evt ) {
        switch( evt.id ) {
            case Event.WINDOW_DESTROY:
                dispose();
                return( true );
            case  Event.ACTION_EVENT:
                if( mainClass.DISMISS.equals( evt.arg ) ) {
                    dispose();
                    return( true );
                }
            }
            return( false );
        }

        public void   setButton( ) {
            Button     b;

            b = new Button( mainClass.DISMISS );
            b.setFont( new Font("TimesRoman", Font.ITALIC, 19) );
            b.setBackground( Color.black );
            b.setForeground( Color.green );
            add("South", b);
        }

        public void   paint( Graphics g ) {
            Dimension     d;
            int           x[] = new int[50];
            int           y[] = new int[50];

            d = size();
            g.setColor( Color.green );
            for( int i = 0; i < x.length-1; i++) {
                x[i] = (int)((d.width-20) * Math.random() + 10);
                y[i] = (int)((d.height-20) * Math.random() + 10);
            }
            x[49] = x[0];
            y[49] = y[0];

            Polygon   p = new Polygon(x, y, 50);

            g.fillPolygon( p );
            g.setColor( Color.black );
            g.setFont( new Font("TimesRoman", Font.PLAIN, 25) );
            g.drawString("Use mouse click in the field that", 15, d.height/4);
            g.drawString("You want to input data", 15, 2*d.height/4);
            g.drawString("Multi-dimensional spatial database V 1.0",15,4*d.height/5);
        }
    };


    /*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
     *
```

```
 *  Class  ReportDialog
 *
 * * * * * * * * * * * * * * * * * * * * * * * * */
class   ReportDialog  extends Dialog {
    mainClass       parent;

    public  ReportDialog( mainClass parent ) {
        super(parent, " Report Error Window ", true);
        setForeground( Color.red );
        setBackground( Color.pink );
        this.parent = parent;

        int             WIDTH = 500;
        int             HEIGHT = 200;
        Dimension       d;

        d = parent.size();
        setButton();
        resize(WIDTH, HEIGHT);
        setResizable( false );
    }

    public  boolean     handleEvent( Event evt ) {
        switch( evt.id ) {
            case    Event.WINDOW_DESTROY:
                dispose();
                return( true );
            case    Event.ACTION_EVENT:
                if( mainClass.DISMISS.equals( evt.arg ) ) {
                    dispose();
                    return( true );
                }
        }
        return( false );
    }

    public  void    setButton( ) {
        Button      b;

        b = new Button( mainClass.DISMISS );
        b.setFont( new Font("TimesRoman", Font.ITALIC, 19) );
        b.setBackground( Color.black );
        b.setForeground( Color.green );
        add("South",  b);
    }
    public  void    paint( Graphics g ) {
        Dimension     d = size();

        g.setColor( Color.black );
        g.setFont( new Font("TimesRoman", Font.PLAIN, 25) );
        g.drawString("If you find something wrongs", 15, d.height/4);
        g.drawString("Please send mail to", 15, 2*d.height/4);
        g.drawString("E-mail asvaser@a.cs.okstate.edu.", 15, 4*d.height/5);
    }
};


/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
 *
 *  Class  AboutDialog
 *
 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
class   AboutDialog   extends Dialog {
    mainClass      parent;

    public  AboutDialog( mainClass parent ) {
        super(parent, " About Window ", true);
        setForeground( Color.red );
        setBackground( Color.pink );
        this.parent = parent;

        int             WIDTH = 435;
        int             HEIGHT = 200;
        Dimension       d;

        d = parent.size();
        setButton();
        resize(WIDTH, HEIGHT);
        setResizable( false );
    }

    public  boolean    handleEvent( Event evt ) {
        switch( evt.id ) {
            case    Event.WINDOW_DESTROY:
                dispose();
                return( true );
            case    Event.ACTION_EVENT:
                if( mainClass.DISMISS.equals( evt.arg ) ) {
                    dispose();
                    return( true );
                }
        }
        return( false );
    }

    public  void    setButton( ) {
        Button      b;

        b = new Button( mainClass.DISMISS );
        b.setFont( new Font("TimesRoman", Font.ITALIC, 19) );
        b.setBackground( Color.black );
        b.setForeground( Color.green );
        add("South",  b);
    }

    public  void    paint( Graphics g ) {
        Dimension       d = size();

        g.setColor( Color.black );
        g.setFont( new Font("System", Font.PLAIN, 25) );
        g.drawString("Multi-dimensional Spatial Database", 15, d.height/4);
        g.drawString(" By Veera A.   Computer Science", 15, 2*d.height/4);
        g.drawString("   Oklahoma State University", 15, 4*d.height/5);
    }
};


/*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
 *
 *  Class  NewDialog
 *
 *  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
class   NewDialog    extends Dialog {
    mainClass       parent;
    Label           dimLabel;
    TextField       dimField;

    public  NewDialog( mainClass parent ) {
        super(parent, " New Input  Window ", true);
        setForeground( Color.red );
        setBackground( Color.pink );
        this.parent = parent;

        int             WIDTH = 500;
    int             HEIGHT = 150;
    Dimension       d;

    d = parent.size();
    displayDataField();
    resize(WIDTH, HEIGHT);
    setResizable( false );
}

public boolean      handleEvent( Event evt ) {
    switch( evt.id ) {
        case    Event.WINDOW_DESTROY:
            dispose();
            return( true );
        case    Event.ACTION_EVENT:
            if( mainClass.OK.equals( evt.arg ) ) {
                updateData();
                dispose();                  // Check some Error here
                return( true );
            }
            if( mainClass.CANCEL.equals( evt.arg ) ) {
                parent.flag = false;
                dispose();
                return( true );
            }
    }
    return( false );
}

public void     updateData( ) {
    try {
        parent.dimension = new Integer( dimField.getText() ).intValue();
        parent.list = new IndexClass( parent.dimension );
        parent.root = parent.list.RTreeNewIndex();
        parent.flag = true;
        parent.FileName = " No File                    ";
    } catch( Exception e ) {
        parent.flag = false;
        parent.showErrorDialog("Dimension is not an Integer number");
        System.err.println( e );
    }
}

public Panel    setButton( ) {
    Panel       p = new Panel();
```

```
        Button      b;

        b = new Button( mainClass.OK );
        b.setFont( new Font("TimesRoman", Font.ITALIC, 19));
        b.setBackground( Color.black );
        b.setForeground( Color.green );
        p.add( b );
        b = new Button( mainClass.CANCEL );
        b.setFont( new Font("TimesRoman", Font.ITALIC, 19));
        b.setBackground( Color.black );
        b.setForeground( Color.red );
        p.add( b );
        return p;
    }

    public  void    addFormComponent(GridBagLayout grid, Component comp,
                                        GridBagConstraints c) {
        grid.setConstraints(comp, c);
        add( comp );
    }

    public  void    displayDataField( ) {
        GridBagLayout          gridbag = new GridBagLayout();
        GridBagConstraints     constraints = new GridBagConstraints();

        dimLabel = new Label("Dimension: ");
        dimField = new TextField( 5 );
        setFont( new Font("TimesRoman", Font.BOLD, 18) );
        setBackground( Color.lightGray );
        setLayout( gridbag );

        constraints.fill = GridBagConstraints.NONE;
        constraints.weighty = 0.0;
        constraints.anchor = GridBagConstraints.WEST;
        addFormComponent(gridbag, dimLabel, constraints);
        constraints.gridwidth = GridBagConstraints.REMAINDER;
        addFormComponent(gridbag, dimField, constraints);

        constraints.anchor = GridBagConstraints.SOUTH;
        addFormComponent(gridbag, setButton(), constraints);
    }
};


/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
```

```
 *
 *  Class  ReadURLDialog
 *
 * * * * * * * * * * * * * * * * * * * * * * * */
class   ReadURLDialog   extends Dialog {
    mainClass       parent;
    Label           URLlabel;
    TextField       URLfield;
    public  ReadURLDialog( mainClass parent ) {
        super(parent, " Input URL File Name  Window ", true);
        setForeground( Color.red );
        setBackground( Color.pink );
        this.parent = parent;

        int             WIDTH = 550;
        int             HEIGHT = 150;
        Dimension       d;

        d = parent.size();
        displayDataField();
        resize(WIDTH, HEIGHT);
        setResizable( false );
    }

    public  boolean     handleEvent( Event evt ) {
        switch( evt.id ) {
            case    Event.WINDOW_DESTROY:
                dispose();
                return( true );
            case    Event.ACTION_EVENT:
                if( mainClass.OK.equals( evt.arg ) ) {
                    updateData();
                    dispose();          // Check some Error here
                    return( true );
                }
                if( mainClass.CANCEL.equals( evt.arg ) ) {
                    parent.flag = false;
                    dispose();
                    return( true );
                }
        }
        return( false );
    }

    public  void    updateData( ) {
```

```
    try {
        parent.FileName = URLfield.getText();
        parent.flag = true;
    } catch( Exception e ) {
        parent.flag = false;
        parent.showErrorDialog("No URL File Name");
        System.err.println( e );
    }
}

public  Panel   setButton( ) {
    Panel       p = new Panel();
    Button      b;

    b = new Button( mainClass.OK );
    b.setFont( new Font("TimesRoman", Font.ITALIC, 19));
    b.setBackground( Color.black );
    b.setForeground( Color.green );
    p.add( b );
    b = new Button( mainClass.CANCEL );
    b.setFont( new Font("TimesRoman", Font.ITALIC, 19));
    b.setBackground( Color.black );
    b.setForeground( Color.red );
    p.add( b );
    return p;
}

public  void    addFormComponent(GridBagLayout grid, Component comp,
                                    GridBagConstraints c) {
    grid.setConstraints(comp, c);
    add( comp );
}

public  void    displayDataField( ) {
    GridBagLayout       gridbag = new GridBagLayout();
    GridBagConstraints  constraints = new GridBagConstraints();

    URLlabel = new Label("URL File Name: ");
    URLfield = new TextField( 40 );
    setFont( new Font("TimesRoman", Font.BOLD, 18) );
    setBackground( Color.lightGray );
    setLayout( gridbag );
    constraints.anchor = GridBagConstraints.WEST;
    constraints.weighty = 1.0;
```

```
        constraints.gridwidth = 1;
        addFormComponent(gridbag, URLlabel, constraints);

        constraints.gridwidth = GridBagConstraints.REMAINDER;
        constraints.fill = GridBagConstraints.HORIZONTAL;
        addFormComponent(gridbag, URLfield, constraints);

        constraints.anchor = GridBagConstraints.SOUTH;
        addFormComponent(gridbag, setButton(), constraints);
    }
};


/* * * * * * * * * * * * * * * * * * * * * * * */
 *
 * Class ErrorDialog
 *
 * * * * * * * * * * * * * * * * * * * * * * * * */
class   ErrorDialog   extends Dialog {
    mainClass      parent;
    String         err;

    public ErrorDialog(mainClass parent, String err) {
        super(parent, " Error Window ", true);
        setForeground( Color.red );
        setBackground( Color.pink );
        this.parent = parent;
        this.err = err;

        int             WIDTH;
        int             HEIGHT = 150;
        Dimension       d;

        d = parent.size();
        setButton();

        WIDTH = err.length() * 15;
        WIDTH = WIDTH < 500 ? 500 : WIDTH;
        resize(WIDTH, HEIGHT);
        setResizable( false );
    }

    public boolean    handleEvent( Event evt ) {
        switch( evt.id ) {
            case    Event.WINDOW_DESTROY:
                dispose();
                return( true );
            case    Event.ACTION_EVENT:
                if( mainClass.DISMISS.equals( evt.arg ) ) {
                    dispose();
                    return( true );
                }
        }
        return( false );
    }

    public  void    setButton( ) {
        Button      b;

        b = new Button( mainClass.DISMISS );
        b.setFont( new Font("TimesRoman", Font.ITALIC, 19) );
        b.setBackground( Color.black );
        b.setForeground( Color.green );
        add("South",  b);
    }

    public  void    paint( Graphics g ) {
        Dimension       d = size();

        g.setColor( Color.black );
        g.setFont( new Font("TimesRoman", Font.PLAIN, 25) );
        g.drawString(err, 10, d.height/2);
    }
};


/*  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
 *
 *  Class  DisplayDialog
 *
 *  *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   */
class   DisplayDialog extends Dialog {
    mainClass       parent;
    String          commandString;
    TextField       rectField[][] = new TextField[2][];
    TextField       recordField;

    private Label       rectLabel[] = new Label[2];
    private Label       dataLabel;
```

```
public  DisplayDialog(mainClass parent, String showString) {
    super(parent, showString + "Rectangle Dimension: " + parent.dimension +
                    " ", true);
    setBackground( Color.lightGray );
    setForeground( Color.black );
    commandString = showString;
    this.parent = parent;

    int  WIDTH = ( (parent.dimension < 10) ? parent.dimension : 10 ) * 100 + 179;
    int  HEIGHT = ( ((parent.dimension - 1) / 10 ) + 1) * 70 + 135;
    Dimension     d;

    d = parent.size();
    WIDTH = (WIDTH < 280) ? 350 : WIDTH;
    makeDataField();
    displayDataField();
    resize(WIDTH, HEIGHT);
    setResizable( false );
}

public  boolean    handleEvent( Event evt ) {
    switch( evt.id ) {
        case    Event.WINDOW_DESTROY:
            dispose();
            return( true );
        case    Event.ACTION_EVENT:
            if( commandString.equals( evt.arg ) ) {
                updateData();
                dispose();            // Check some Error here
                return( true );
            }
            if( mainClass.CANCEL.equals( evt.arg ) ) {
                parent.flag = false;
                dispose();
                return( true );
            }
    }
    return( false );
}

public  Panel   setButton( ) {
    Panel       p = new Panel();
    Button      b;

    b = new Button( commandString );
    b.setFont( new Font("TimesRoman". Font.ITALIC, 19));
    b.setBackground( Color.black );
    b.setForeground( Color.green );
    p.add( b ):
    b = new Button( mainClass.CANCEL );
    b.setFont( new Font("TimesRoman", Font.ITALIC, 19));
    b.setBackground( Color.black );
    b.setForeground( Color.red );
    p.add( b );
    return p;
}

public void    makeDataField( ) {
    parent.rectField = new float[parent.dimension * 2];
    parent.recordField = new String();
    dataLabel = new Label("RECORD: ");
    recordField = new TextField( 20 );
    recordField.setBackground( Color.white );
    recordField.setForeground( Color.black );
    for( int i = 0; i < rectField.length; i++ )
        rectField[i] = new TextField[parent.dimension];
    for( int i = 0; i < rectField.length; i++ )
        for( int j = 0; j < rectField[i].length; j ++ )
            rectField[i][j] = new TextField( 9 );
}

public void    updateData( ) {
    try {
        for( int i = 0; i < rectField[0].length; i++ ) {
            parent.rectField[i] = new Float( rectField[0][i].getText() ).floatValue();
            parent.rectField[i + parent.dimension] =
                            new Float( rectField[1][i].getText() ).floatValue();
        }
        if( commandString.equals( mainClass.SEARCH ) )
            parent.recordField = new String();
        else
            parent.recordField = new String( recordField.getText() );
        parent.flag = true;
    } catch( Exception e ) {
        parent.flag = false;
        parent.showErrorDialog("Some fields were wrong in data field");
        System.err.println( e );
    }
}
```

```
public void    makeLabel( ) {
    rectLabel[0] = new Label("Coordinate MIN: ");
    rectLabel[1] = new Label("Coordinate MAX: ");
}

public void    addFormComponent(GridBagLayout grid, Component comp,
                                            GridBagConstraints c) {
    grid.setConstraints(comp, c);
    add( comp );
}

public Color  getBackColor( ) {
    Color   tempColor = Color.lightGray;         // Initilized for INSERT

    if( commandString.equals( mainClass.DELETE ) )
        tempColor = Color.pink;
    else
        if( commandString.equals( mainClass.SEARCH ) )
            tempColor = Color.yellow;
    return  tempColor;
}

public synchronized  void   displayDataField( ) {
    GridBagLayout          gridbag = new GridBagLayout();
    GridBagConstraints     constraints = new GridBagConstraints();
    Color                  foreColor[] = new Color[2];
    Color                  backColor[] = new Color[2];
    int                    i, j, loop;

    setFont( new Font("TimesRoman", Font.BOLD, 18) );
    setBackground( getBackColor() );
    setLayout( gridbag );

    constraints.fill = GridBagConstraints.NONE;
    constraints.weighty = 0.0;
    if( !commandString.equals( mainClass.SEARCH ) ) {
        constraints.anchor = GridBagConstraints.NORTHWEST;
        addFormComponent(gridbag, dataLabel, constraints);
        constraints.gridwidth = GridBagConstraints.REMAINDER;
        addFormComponent(gridbag, recordField, constraints);
    }

    constraints.anchor = GridBagConstraints.WEST;

    foreColor[0] = Color.white;
```

```
        backColor[0] = Color.black;
        foreColor[1] = Color.green ;
        backColor[1] = Color.darkGray;
        j = loop = 0;
        do {
            makeLabel();
            for( i = 0; i < rectLabel.length; i++ ) {
                constraints.weightx = 0.0;
                constraints.gridwidth = 1;
                addFormComponent(gridbag, rectLabel[i], constraints);
                for( j = loop*10; j < rectField[i].length - 1  &&  j < loop*10 + 9; j++ ) {
                    rectField[i][j].setForeground( foreColor[loop % 2] );
                    rectField[i][j].setBackground( backColor[loop % 2] );
                    addFormComponent(gridbag, rectField[i][j], constraints);
                }
                constraints.weightx = 1.0;
                constraints.gridwidth = GridBagConstraints.REMAINDER;
                rectField[i][j].setForeground( foreColor[loop % 2] );
                rectField[i][j].setBackground( backColor[loop % 2] );
                addFormComponent(gridbag, rectField[i][j], constraints);
            }
            loop++;
        } while( j < rectField[0].length - 1 );

        constraints.anchor = GridBagConstraints.SOUTH;
        addFormComponent(gridbag, setButton(), constraints);
    }
};


/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
 *
 *  Class  DisplaySearchDialog
 *
 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
class   DisplaySearchDialog  extends Dialog {
    mainClass       parent;
    TextField       rectField[][] = new TextField[2][];
    TextField       recordField;

    private Label   rectLabel[] = new Label[2];
    private Label   dataLabel;

    public  DisplaySearchDialog( mainClass parent ) {
        super(parent, " Search Dimension: " + parent.dimension + " ", true);
    setBackground( Color.lightGray );
    setForeground( Color.black );
    this.parent = parent;

    int  WIDTH = ( (parent.dimension < 10) ? parent.dimension : 10 ) * 100 + 179;
    int  HEIGHT = ( ((parent.dimension - 1) / 10 ) + 1) * 70 + 135;
    Dimension     d;

    d = parent.size();
    WIDTH = (WIDTH < 280) ? 350 : WIDTH;
    makeDataField();
    displayDataField();
    resize(WIDTH, HEIGHT);
    setResizable( false );
}

public  boolean     handleEvent( Event evt ) {
    switch( evt.id ) {
        case    Event.WINDOW_DESTROY:
            dispose();
            return( true );
        case    Event.ACTION_EVENT:
            if( mainClass.DISMISS.equals( evt.arg ) ) {
                dispose();
                return( true );
            }
    }
    return( false );
}

public  Panel   setButton( ) {
    Panel       p = new Panel();
    Button      b;

    b = new Button( mainClass.DISMISS );
    b.setFont( new Font("TimesRoman", Font.ITALIC, 19) );
    b.setBackground( Color.black );
    b.setForeground( Color.green );
    p.add( b );
    return  p;
}

public  void    makeDataField( ) {
    dataLabel = new Label("RECORD: ");
    recordField = new TextField(parent.recordField, 20);
```

```
        recordField.setEditable( false );
        recordField.setBackground( Color.white );
        recordField.setForeground( Color.black );
        for( int i = 0; i < rectField.length; i++ )
            rectField[i] = new TextField[parent.dimension];
        for( int i = 0; i < rectField[0].length; i++ ) {
            rectField[0][i] = new TextField( Float.toString( parent.rectField[i] ), 9);
            rectField[0][i].setEditable( false );
            rectField[1][i] = new TextField(
            Float.toString( parent.rectField[i+parent.dimension] ), 9);
            rectField[1][i].setEditable( false );
        }
    }

    public void    makeLabel( ) {
        rectLabel[0] = new Label("Coordinate MIN: ");
        rectLabel[1] = new Label("Coordinate MAX: ");
    }

    public void  addFormComponent(GridBagLayout grid, Component comp,
                                          GridBagConstraints c) {
        grid.setConstraints(comp, c);
        add( comp );
    }

    public void    displayDataField( ) {
        GridBagLayout           gridbag = new GridBagLayout();
        GridBagConstraints      constraints = new GridBagConstraints();
        Color                   foreColor[] = new Color[2];
        Color                   backColor[] = new Color[2];
        int                     i, j, loop;

        setFont( new Font("TimesRoman", Font.BOLD, 18) );
        setBackground( Color.yellow );
        setLayout( gridbag );

        constraints.fill = GridBagConstraints.NONE;
        constraints.weighty = 0.0;
        constraints.anchor = GridBagConstraints.NORTHWEST;
        addFormComponent(gridbag, dataLabel, constraints);
        constraints.gridwidth = GridBagConstraints.REMAINDER;
        addFormComponent(gridbag, recordField, constraints);

        constraints.anchor = GridBagConstraints.WEST;
        foreColor[0] = Color.white;
        backColor[0] = Color.black;
        foreColor[1] = Color.green ;
        backColor[1] = Color.darkGray;
        j = loop = 0;
        do {
            makeLabel();
            for( i = 0; i < rectLabel.length; i++ ) {
                constraints.weightx = 0.0;
                constraints.gridwidth = 1;
                addFormComponent(gridbag, rectLabel[i], constraints);
                for( j = loop*10; j < rectField[i].length - 1  &&  j < loop*10 + 9; j++ ) {
                    rectField[i][j].setForeground( foreColor[loop % 2] );
                    rectField[i][j].setBackground( backColor[loop % 2] );
                    addFormComponent(gridbag, rectField[i][j], constraints);
                }
                constraints.weightx = 1.0;
                constraints.gridwidth = GridBagConstraints.REMAINDER;
                rectField[i][j].setForeground( foreColor[loop % 2] );
                rectField[i][j].setBackground( backColor[loop % 2] );
                addFormComponent(gridbag, rectField[i][j], constraints);
            }
            loop++;
        } while( j < rectField[0].length - 1 );

        constraints.anchor = GridBagConstraints.SOUTH;
        addFormComponent(gridbag, setButton(), constraints);
    }
};


/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
 *
 *  Class  ReadFileURL
 *
 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
class   ReadFileURL {
    mainClass       parent;
    StringBuffer    sb;

    public  ReadFileURL(mainClass parent, URL location)
                                    throws MalformedURLException, IOException {
        sb = new StringBuffer();
        this.parent = parent;
    int     loop;

    try {
        InputStream is = location.openStream();

        int     oneChar;

        while( (oneChar = is.read()) != -1 )
            sb.append( (char)oneChar );
        is.close();
        parent.dimension = readInt( 0 );
        parent.list = new IndexClass( parent.dimension );
        parent.root = parent.list.RTreeNewIndex();
        loop = 4;
        while( loop < sb.length() ) {
            loop = ReadRect( loop );
            parent.r = new RectClass(parent.rectField, parent.recordField);
            parent.root=parent.list.RTreeInsertRect(parent.r, parent.root, 0);
        }
    } catch( IOException e ) {
        parent.showErrorDialog( e.toString() );
    }
}

private final int    readInt( int i )       throws IOException {
    int     ch1 = sb.charAt( i++ );
    int     ch2 = sb.charAt( i++ );
    int     ch3 = sb.charAt( i++ );
    int     ch4 = sb.charAt( i++ );

    return( (ch1 << 24) + (ch2 << 16) + (ch3 << 8) + (ch4 << 0) );
}

private final float  readFloat( int i )   throws IOException {
    return  Float.intBitsToFloat( readInt( i ) );
}

/**
 * Reads a sub array as a sequence of bytes.
 * @param b the data to be written
 * @param off the start offset in the data
 * @param len the number of bytes that are written
 * @exception IOException If an I/O error has occurred.
 */
private native int   readBytes(byte b[], int off, int len)    throws IOException;
```

```
    /**
     * Reads data into an array of bytes.  This method blocks
     * until some input is available.
    * @return the actual number of bytes read, -1 is
    *        returned when the end of the stream is reached.
    * @exception IOException If an I/O error has occurred.
    */
    public int   read(byte b[])  throws IOException {
        return readBytes(b, 0, b.length);
    }

    private final int     ReadRect( int j ) {
        char          temp[];
        int           size;

        try {
            parent.rectField = new float[parent.dimension * 2];
            for( int i = 0; i < parent.rectField.length; i++, j += 4 )
                parent.rectField[i] = readFloat( j );
            size = readInt( j );
            j += 4;            // size of integer variable
            temp = new char[size];
            sb.getChars(j, j+size, temp, 0);
            parent.recordField = new String( temp );
            j += size;        // size of Record variable
        } catch( IOException e ) {
            System.err.println( e );
        }
        return j;
    }
};
```

VITA

Veera Asvasermcharoen

Candidate for the Degree of

Master of Science

Thesis: DESIGN AND IMPLEMENTATION OF A SPATIAL INDEX STRUCTURE FOR A SPATIAL DATABASE WITH JAVA

Major Field: Computer Science

Biographical:

Personal Data: Born in Bangkok, Thailand, January, 1969, the son of Pronchai and Chanida Asvasermcharoen.

Education: Graduated from Assumption College, Bangkok, Thailand in March 1987; received Bachelor of Science of Science degree in Applied Statistics from King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand in April 1991. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December, 1996.

Experience: Programmer in the Siam Cement CO., Ltd., Bangkok, Thailand, April 1991 to October 1991. Programmer in the Bangkok Bank Limited, Head Office, Bangkok, Thailand, October 1991 to December 1993.