Software development as applied to Java Create Report project

Giap Thai
The University of Montana

Recommended Citation
Thai, Giap, "Software development as applied to Java Create Report project" (2002). Graduate Student Theses, Dissertations, & Professional Papers. 5124.
https://scholarworks.umt.edu/etd/5124
Permission is granted by the author to reproduce this material in its entirety, provided that this material is used for scholarly purposes and is properly cited in published works and reports.

**Please check "Yes" or "No" and provide signature**

Yes, I grant permission

No, I do not grant permission

Author's Signature: 

Date: 12/20/2002

Any copying for commercial purposes or financial gain may be undertaken only with the author's explicit consent.
Software Development as Applied to Java

Create Report Project

by

Giap Thai

B.E. The Royal Melbourne Institute of Technology, Australia, 1999

Presented in partial fulfillment of the requirements

for the degree of

Master of Computer Science

Department of Computer Science, The University of Montana

December, 2002

Approved by

Chairperson

Dean, Graduate School

12-23-02

Date
The paper describes the software development approach of implementing the Java Create Report web application for Education Logistics, Inc. The Java Create Report software allows the naive user, without any knowledge of Structured Programming Language (SQL), to create a query and easily customize it to generate well-formatted reports as desired. During the software requirements and specification process, the customer’s needs are evaluated. Choices of implementation languages, database drivers and unanticipated requirements have been taken into consideration. In the typical use case presented in this document, the user can create a new query, define filters, output fields and the query sort order, then run the query. This use case is analyzed in the software design. The Java Create Report software employs the three-tier database architecture. The client applet implements the software Graphic User Interface, handles user events and manages user queries. The server servlet implementation performs user query transactions. The Edulog Microsoft Foxpro database could be located on the same server machine as the server servlet or located on a different machine. To ensure source code maintenance and reusability, the coding standard conforms to Edulog programming coding standards. The software resource management uses a number of property files that are bundled in a Java Archive Resource (JAR) file for configuration flexibility. Multiple testing methods that ensure the Java Create Report software quality are described. Finally, a recommendation for future enhancement of the software design and functionality has been suggested.
Acknowledgement

I would like to thank my advisor, Professor Alden Wright, for his constructive criticism, his great help and advises that made it possible to complete this paper.

I would like to thank the contributions of the committee members, Dr. Joel Henry from the Computer Science department and Dr. Greg Cripe from the Mathematical Sciences department.

The help of many other individuals from Education Logistics, Inc are gratefully acknowledged.
## Contents

1. Background of Java Create Report .......................................................... 1

2. Project Requirements and Specifications ................................................. 2
   2.1. Evaluation on Customer's Need ........................................................... 2
   2.2. Planning for Unanticipated Requirement Features ............................ 6
   2.3. Choice of Implementation Language ............................................... 6
   2.4. Choice of Database Driver ............................................................... 9
   2.5. Requirements Documentation ......................................................... 10

3. Use Case Analysis ..................................................................................... 10
   3.1. Use Case for All the Important Ways in Which a User Might Use JCR ... 10
   3.2. A Detailed Use Case Analysis for One Use Case, Namely the Use Case
       Where the User Creates a Query, Defines Filters for the Query, Defines
       Output Fields and Sort Order for the Query and then Runs the Query .... 14
       3.2.1. Description .................................................................................. 14
       3.2.2. Typical Course of Events .......................................................... 14
       3.2.3. Associated Sequence Diagram .................................................. 16

4. High-Level Design ..................................................................................... 16
   4.1. Functional Unit Diagram of the JCR Three-Tier Database Architecture ... 16
   4.2. Choice of Class Name and Method Name .......................................... 17

5. Detailed Design ......................................................................................... 17
   5.1. Class and Resource List ..................................................................... 19
   5.2. Class Details ..................................................................................... 22
   5.3. JCR Client and Server Design ........................................................... 22
       5.3.1. JCR Client Design ................................................................. 22
       5.3.2. JCR Server Design ............................................................... 28
   5.4. Description of the Initialization Process ............................................ 30
   5.5. Software Document ........................................................................... 33

6. Code Development .................................................................................... 33
   6.1. Choice of Programming Tool and Environment ................................. 33
   6.2. Coding Standard ............................................................................... 34
   6.3. Program Resource Management ....................................................... 36

7. Testing ....................................................................................................... 37
   7.1. Class Testing ..................................................................................... 37
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.</td>
<td>Functional Testing</td>
<td>39</td>
</tr>
<tr>
<td>7.3.</td>
<td>Edulog Testing Procedure</td>
<td>41</td>
</tr>
<tr>
<td>8.</td>
<td>Conclusion</td>
<td>42</td>
</tr>
<tr>
<td>9.</td>
<td>Bibliography</td>
<td>44</td>
</tr>
<tr>
<td>10.</td>
<td>Appendix A: JCR Requirement Document</td>
<td>45</td>
</tr>
<tr>
<td>11.</td>
<td>Appendix B: Associated Sequence Diagram for the Use Case Where the User Creates a Query, Defines Filters for the Query, Defines Output Fields and Sort Order for the Query and Finally Runs the Query</td>
<td>54</td>
</tr>
<tr>
<td>12.</td>
<td>Appendix C: JCR Client Applet Classes</td>
<td>57</td>
</tr>
<tr>
<td>13.</td>
<td>Appendix D: JCR Server Servlet Classes</td>
<td>66</td>
</tr>
<tr>
<td>14.</td>
<td>Appendix E: Serialized Classes that are Shared Between JCR Client Applet and JCR Server Servlet</td>
<td>69</td>
</tr>
<tr>
<td>15.</td>
<td>Appendix F: Client Class Diagram</td>
<td>73</td>
</tr>
<tr>
<td>16.</td>
<td>Appendix G: Server Class Diagram</td>
<td>76</td>
</tr>
<tr>
<td>17.</td>
<td>Appendix H: Software Document</td>
<td>77</td>
</tr>
<tr>
<td>18.</td>
<td>Appendix I: Edulog Coding Standards in Java Context</td>
<td>96</td>
</tr>
</tbody>
</table>
1. **Background of Java Create Report**

Edulog (Education Logistics, Inc) has been providing its customers with user-defined report software which was developed in Foxpro and using Query Maker - a third party Foxpro package. This software however requires a complex installation of the Foxpro run time environment, application database, the third party Query Maker package and the application itself for each user's machine. As a result, the software has a high cost of maintenance and customer support from the company.

Therefore, Edulog proposed a new version of user-defined report software, which will eliminate the high cost of maintenance and customer support. The user-defined report software will allow user to easily create a query and customize it to generate well-formatted reports as desired. Using the user-defined report software will be simple and intuitive. The user, without any programming knowledge of Structure Query Language (SQL), will be able to define a query that makes a detailed report of requested output fields and customized information. The user will also be able to reuse queries that were created by other users to directly generate reports or to copy the queries into new queries and customize them to generate desired reports. The software will prevent the user from generating syntactically incorrect and meaningless SQL queries.

After discussion and evaluation Edulog decided that web-based client server software with three tiers database architecture is best suited for the requirements. At the beginning
of 2001 Edulog proposed the JCR (Java Create Report) project. I had an opportunity to undertake the Java Create Report project with Edulog under supervision of Professor Alden Wright.

2. **Project requirements and specifications.**

2.1. **Evaluation on customer’s needs.**

Edulog desires the software to exhibit the following characteristics:

1. The software allows the user to define his/her own query.
2. The software allows the user to reuse his/her old queries or other user queries.
3. The software allows the user to modify existing queries by modifying query fields, query filters and the query sort order.
4. The user should be able to obtain a hard copy of the result of a query.
5. The software must be able to use the existing Edulog Foxpro database.

Other characteristics, which are desired but not essential:

1. Fast retrieval of data from the database
2. Easy integration with existing software ePTS (Electronic Pupil Transportation System) and eFMS (Electronic Facility Maintenance System).
The user should be able to define his/her own query. This means that JCR should allow the user to customize his/her SQL (Structured Query Language) to generate the results of the query as a report. The SQL query could be a simple query such as a query with default output fields, sort order and without filter conditions or a complex query with many output fields, sort order and with many customized filter conditions.

An example for a simple query could be: "SELECT * FROM studnt".

An example for a complex query could be: "SELECT studnt.lastname, studnt.firstname, studnt.school, studnt.grade, studnt.location FROM studnt INNER JOIN stops ON studnt.studntid = stop.stopid WHERE studnt.lastname = 'SMITH' OR studnt.lastname = 'EL WOOD' AND studnt.edulogid >= 2000 AND studnt.state LIKE 'MT' AND studnt.grade = '24' AND NOT(studnt.school) = NULL ORDER BY studnt.lastname, studnt.firstname, studnt.school, studnt.grade"

The user should also be able to use his/her own queries or other user queries. These queries should be available to the user every time that the user uses JCR.

The user could also reuse an existing defined query by copying it into a new query and then modifying the new query to generate a desired report.

JCR should allow the user to modify existing queries. There would be two types of existing queries. The first type should be those queries that had already been defined by the user from the last time user used JCR or those queries that had already been defined
by other users before user starts JCR. The second type should be those queries that have just been created by the user after login.

A user should be able to change a query name so that it is more meaningful to him/her. The user should also be able to modify output fields, filter conditions and sort in order to construct a query to accomplish a new task.

JCR will also have to provide the user with a capability to print out his/her query result. The printing should be presented in a report format for easy reading. The user should be able to choose which information fields are included the printing and how these fields are sorted. The printing capability should not require the user to install an extra print driver.

JCR will allow the user to use several existing Edulog Foxpro databases. Existing Edulog Foxpro databases that are required to interface with are the EdulogNT Foxpro database, the FMS (Facility Management System) Foxpro database, the RMS (Record Management System) Foxpro database. The EdulogNT Foxpro database is the database that has been used by Edulog’s EdulogNT software that is the software for pupil transportation management, attendance boundary planning, and enrollment forecasting. The FMS Foxpro database is the database for facility management that has been used by the Edulog FMS group’s software. The RMS Foxpro database is the database for police record management that has been used by the Logistics RMS group’s software.
Regarding the fast retrieval of data from database requirement, JCR will aim to retrieve data from the database quickly. The speed of data retrieval from the database with JCR will depend on three factors: the amount of data retrieved, the processing time of the query, and the user’s network connection. A user-defined query could have a query result of one record or several ten thousands of records. The number of records in a query depends on the query filter conditions: very restricted query filter conditions will result in a query result containing a few records; less restricted query filter conditions will result in a large query result. The number of requested data fields of a record depends on the number of output fields that user selected when the user defined the query. The user might request report information on one data field or might request on all available data fields. A simple query takes less than a millisecond to process. A more complex query takes more time to process at the web server. However, the processing time of the query will often be very insignificant in comparison to the network transfer time. Users are often from district schools, district transportation departments or district police departments. Some users might use JCR from home. Therefore, user’s network connection varies. It could be anywhere from 56 Kbps for a standard home dial-up connection to a 100 Mbps Ethernet connection. With 100 Mbs connection, transfer several thousand records could take less than a second; with a 56 Kbps connection, transfer of several thousand records could take up to several minutes, which could be a problem for users. In order to deal with this problem, the software will allow user to easily configure the limitation on number of number of records returned by a query. The default value of this limitation number is 1000 records.
JCR is required to easily integrate with Edulog software packages such as ePTS and eFMS. JCR is also required to use existing Edulog Foxpro databases includes EdulogNT Foxpro database and FMS Foxpro database.

2.2. **Planning for unanticipated requirement features**

Desirable features that go beyond the essential features described above had not been defined at the time that the project was proposed. I have discussed these features with Professor Wright and Edulog managers.

It may be necessary to add some additional features after some of JCR features have been built. To prepare for unanticipated features, JCR should be designed to be flexible, which allows adding new features and easily modifying existing features. JCR should also be designed with minimal hard coding. The software constants and resources should be able to be changed by modifying configuration files without recompiling source codes.

2.3. **Choice of implementation language.**

The Edulog proposed architecture for JCR is a Java applet-server application: the applet will display GUIs to user and any request by user will be handled by the applet and then the applet sends these requests to the server to process. The server will examine request from applet and query information from the database. The resulting information will be sent back from the server to the applet to display to the user.
With the requirements that Edulog proposed for JCR, there are several options on implementation language such as:

- Java for applet and Java for server.
- Java for applet and Jsp for server.
- Java for applet and Perl or PHP for server.

Perl is a popular script that is used for general purposed programming invented in 1987 by Larry Wall. It derives from C programming language and to some extents from sed, awk, and the Unix shell. Similarly, PHP is also used for general purposed server-side scripting programming. PHP syntax draws from C, Java and Perl and is relatively easier to learn compared to Perl. PHP has good support for database. Web applications that employ PHP and MySQL database are currently popular. Though Java, Perl and PHP all support Http, communication between Java applet and Perl or PHP server could be fairly complicated. To implement a request from a Java applet to a Perl or PHP server, or a query response from a Perl or PHP server back to a Java applet, a data object must be defined in such a way that both the Java applet and the Perl or PHP server could both support this data object.

Java Server Pages (JSP) is an extension of Java Technology that allows developers easily implement information-rich, dynamic web pages. JSP uses XML-like tags that encapsulate the logic that generates the web page. JSP also employs application logic that can reside in the server as Java Bean. Communication between a Java applet and a JSP
server is also difficult to design as JSP is designed to generate static and dynamic web pages that are sent back to user’s web browser on a user’s request rather than to generate a Java object that can be passed between the JSP server and a Java applet.

Servlet is also another extension of Java Technology that fits seamlessly into a web server framework and can be used to extend the capabilities of the Web server with minimal overhead and maintenance. Unlike Perl and PHP, servlet involves no platform-specific consideration or modifications. Java servlet technology allows a fast transfer of information data between a Java applet and the web server. A suitably-defined Java serialized data structure object will be able to hold data that encapsulates requests from Java applet to Java servlet or a query result sent from servlet to client applet. Edulog also desires to use the Java servlet since this would facilitate in source code maintenance and integration with other Edulog software that have already employed Java applet and Java servlet architecture.

Writing JCR in architecture of Java applet and Java servlet is an advantage for me as well since I have been participating in writing several software packages for Edulog in Java with this architecture such as ePTS (since 1999), Web Standard Report (since 2000), eFMS (since 2000) and Web Bus Locator (since 2002).

Edulog requires JCR to have sophisticated graphical GUIs, therefore the Java Runtime Environment 1.2 (JRE 1.2) or above is required in JCR. Most of the current operating systems have supported JRE 1.0 but not JRE 1.2. Microsoft Window XP and Internet
Explorer 6.0 even disables Java by default, user has to configure his or her browser to allow applet to be loaded, then user still has to download JRE and install JRE to be able to see applet. Therefore, to ensure user to have JRE 1.2, at the first time when user points his or her browser to JCR, JCR will detect to see whether the user’s computer has already installed an appropriate JRE or had a suitable version of JRE. If not, JCR will connect user to Sun homepage (http://Java.sun.com) and ask the user to upgrade or install the latest version of JRE.

2.4. **Choice of database drivers.**

As Java applet and Java servlet architecture is employed in JCR, Java servlet is required to interface with Edulog existing databases. There is several options for database drivers, which will be used for the servlet at the server:

1. Using a third party Java database driver, which is different from JDBC (Java Database Connectivity).
2. Using JDBC 1.1.
3. Using JDBC 1.2.

Using a third party database driver will give more features and capabilities in manipulating Edulog Foxpro databases from JCR. However, the company has to purchase it. Using JDBC 1.2 will give JCR more capability in manipulating the Foxpro database than JDBC 1.1 does. However, third party and JDBC 1.2 database drivers are not compatible with current Edulog software. Therefore, if the third party or JDBC 1.2
database drivers are employed, JCR software will have to reside in a separate server from
the server in which some Edulog softwares that use JDBC 1.1 database drivers resides.
This will result in extra cost of purchase new server and maintenance. Therefore, JDBC
1.1 - a free database driver from Sun - is chosen for JCR software.

2.5. **Requirements documentation.**

JCR Requirements and specifications have been discussed between professor Wright,
Edulog managers, Victor Valenti who is in charge of documentation from Edulog and I.
Victor has prepared the requirements for JCR that specifies features and functionalities
for JCR. This document is included in appendix A.

3. **Use case analysis.**

3.1. **Use cases for all the important ways in which a user might use JCR.**

The following use cases describe all of the important ways in which a user might use
JCR:

a. User creates a new query:

1. User clicks on the “Query” button to go to the query list panel
2. User enters a name for the query on query name text field
3. User clicks on “Create Query” button to create a new query.
b. User copies a query:
   1. User clicks on “Query” button to go to the query list panel
   2. User selects an existing query
   3. User enters a name for the query on query name text field
   4. User clicks on “Copy Query” button to copy the selected query.

c. User renames a query:
   1. User clicks on “Query” button to go to the query list panel.
   2. User selects an existing query
   3. User enters a new name for the query on query name text field
   4. User clicks on “Rename Query” button to rename the selected query.

d. User deletes a query:
   1. User clicks on “Query” button to go to the query list panel.
   2. User selects an existing query.
   3. User clicks on “Delete Query” button to delete selected query.

e. User queries all of the queries available on the server:
   1. User clicks on “Query” button to go to the query list panel
   2. User clicks on “Refresh Query” button to request all the available queries from the server.
f. User creates a new query and defines filters for the query:

1. User clicks on “Query” button to go to the query list panel
2. User enters a name for the query on the query name text field
3. User clicks on “Create Query” to create a new query.
4. User clicks on “Enter Record Filter” to go to the filter panel.
5. User selects a filter field from the “Field” drop-down box
6. User selects a filter comparison operator from the “Comparison” drop-down box.
7. Users enter a value into the “Value” field -> Users selects “Add Filter” button to add this filter.

g. User creates a new query and defines output fields and sort order for the query:

1. User clicks on “Query” button to go to the query list panel
2. User enters a name for the query on the query name text field.
3. User clicks on “Create Query” to create a new query.
4. User clicks on “Output/Sort Order” button to go to the output(sort order panel
5. User selects output fields from the “Select Output Field” list
6. User selects sort order by clicking on “Ascending” or “Descending” radio button
7. User clicks on “Save Output Fields & Sort Order” button.
h. User creates a new query and defines filters for the query, defines output fields and sort order for the query, then run the query:

1. User clicks on "Query" button to go to the query panel
2. User enters a name for the query on the query name text field.
3. User clicks on "Create Query" button to create a new query
4. User clicks on "Enter Record Filters" to go to the query filter panel
5. User selects a filter field from the filter "Field" drop-down box
6. User selects a filter comparison from the filter "Comparison" drop-down box.
7. User enters a value in to the filter "Value" field
8. User click on "Add Filter" button
9. User clicks on "Output/Sort Order" button to go to the output and sort order panel
10. User selects output fields from the "Select Output Fields" list
11. User selects sort order by clicking on "Ascending" or "Descending" radio button
12. User clicks on "Save Output Fields & Sort Order".
13. User clicks on "Run Query" button to run the query.

i. User runs a query:

1. User clicks on "Query" to go to the query panel
2. User select an existing query -> User click on “Run Query” button to run the selected query.

3.2. A detailed use case analysis for one use case, namely the use case where the user creates a query, defines filters for the query, defines output fields and sort order for the query and then runs the query.

3.2.1. Description.

Actor: A JCR user.

Purpose: Create and customize a query to accomplish a new task.

Overview: A user starts JCR, creates a new query, defines a desired filter, then defines the desired output fields and selects a sort order.

3.2.2. Typical course of events.
<table>
<thead>
<tr>
<th>Actor Action</th>
<th>JCR Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. User starts JCR</td>
<td>2. JCR loads all the existing queries from the database to the user machine.</td>
</tr>
<tr>
<td>3. User selects the “Query” button.</td>
<td>4. JCR displays the “Query Console” panel.</td>
</tr>
<tr>
<td>5. User enters a query name on the query name field.</td>
<td>7. JCR creates a new query with user-entered name and default filters, output fields and sort order and saves into the database</td>
</tr>
<tr>
<td>6. User selects the “Create Query” button.</td>
<td>9. JCR displays the “Record Filter” panel.</td>
</tr>
<tr>
<td>8. User selects the “Enter Record Filter” button.</td>
<td>10. User selects a filter field.</td>
</tr>
<tr>
<td>10. User selects a filter field.</td>
<td>11. User selects a filter operator.</td>
</tr>
<tr>
<td>12. User enters a value to filter value field.</td>
<td>13. User selects the “Add Filter” button.</td>
</tr>
<tr>
<td>13. User selects the “Add Filter” button.</td>
<td>15. User selects the “Output/Sort order” button.</td>
</tr>
<tr>
<td>15. User selects the “Output/Sort order” button.</td>
<td>17. User selects output fields from the output field list.</td>
</tr>
<tr>
<td>17. User selects output fields from the output field list.</td>
<td>18. User selects a sort order from the “ASC/DESC SORT” radio buttons.</td>
</tr>
<tr>
<td>18. User selects a sort order from the “ASC/DESC SORT” radio buttons.</td>
<td>19. User selects the “Save Output Fields &amp; Sort Order” button.</td>
</tr>
<tr>
<td>19. User selects the “Save Output Fields &amp; Sort Order” button.</td>
<td>20. JCR saves the selected output fields and sort orders into the database for the query.</td>
</tr>
<tr>
<td>21. User clicks on the “Run Query” button.</td>
<td>22. JCR executes the query and displays the result of the query in the “Custom Report” panel.</td>
</tr>
</tbody>
</table>
3.2.3. **Associated Sequence diagrams.**

This document is included in the Appendix B.

4. **High – level design.**

4.1. **Functional unit diagram of the JCR three-tier database architecture.**

The JCR client applet is implemented to provide user with the Graphic User Interface and instructions of how to use the user-defined report software and to handle user’s events as result of user interacting with the software. The JCR client applet will respond with appropriate actions corresponding to user’s events. Upon user’s requests JCR client applet will issue query’s commands as SELECT, UPDATE, INSERT or DELETE that are sent to the JCR server servlet. The JCR server servlet functions are to analyze the JCR client applet’s request and to perform appropriate database query transactions. The query result retrieved from the database is also sent back to the JCR client applet to display to user by the JCR server servlet. The JCR software three-tier design can be seen as the below diagram:
* The JCR client applet sends a request (create query, modify query, run query, delete query) by user to the JCR servlet at the server, and then the servlet performs appropriate SQL operations (insert, update, select, delete) to the Microsoft Foxpro Database to accomplish a task. The Microsoft Foxpro Database could either be located on the same server machine with the server servlet or located on a different server machine.

4.2. **Choices of class names and method names.**

Though it takes more time in typing for long class name and method name, it is useful to have a descriptive name because descriptive class name and method name are easy to understand.

As codes have to conform to Edulog coding standard: the initial letter of the class name will be upper case, for example: QueryManager, IntroPanel, CustomReportPanel, FilterPanel. However, the initial letter of the class method will be lower case, example: runQuery, deleteQuery, insertQuery, updateQuery.

5. **Detailed Design.**

The JCR software design basically consists of three major types of components: the classes that display Graphic User Interface (GUI) and handle user’s events; the classes that manage queries; and the classes that handle communication between JCR client applet and JCR server servlet.
User queries are stored in a Microsoft Visual Foxpro table named “Udr_querylist.dbf”.

When the user creates a new query, this query information of query id, query description, query fields, query filters and query sort order is inserted into the “Udr_querylist.dbf”.

The method that performs this insertion of the query is synchronized so that the concurrent situation where many users create a new query at the same time can be handled. The synchronization process is also implemented for methods that perform “user delete query” action and “user update query” action.

Since I have to write the paper and the class diagrams on the JCR software, I now understand JCR software design much better than when I wrote it last year. The type of JCR software design is called “Singleton” style, that means there is only one instance of all the main classes that are used in the software, for example: there is only one instance of the ImageManager class, only one instance of the QueryManager class, only one instance of the CommManagerGroup class, only one instance of the IniBundle class … in the JCR design. This is convenient in the way of passing reference of class instance, for example, if a class needs a reference of the ImageManager class, since there is only one instance of this class is used through the entire JCR, the reference of the ImageManager class in the UsrDefnRepApplet class can be obtained by declaring the variable that refers to the ImageManager class as “static” and implementing an assessor method for this variable then using “UsrDefnRepApplet.getImageManager()” call to return the reference of the ImageManager class. The normal way of passing instance reference between classes is through class constructor that is not as nice. I have also realized that there are a
number of duplicate utility methods that I should had been able to simplify by moving them into one utility class to avoid the duplication. An example of this problem in the current JCR design is the duplication of the utility method “private AbstractButton buttonProps(AbstractButton,Color)” that appears in three classes QueryPanel, RecordFilterPanel and UsrDefnRepGUIManager.

5.1. **Class and resource list.**

The classes, resource files and their short descriptions are listed as below.

**Client applet classes:**

1. UsrDefnRepApplet - JCR client applet
2. UsrDefnRepGUIManager - JCR main GUI
3. UsrDefnRepInterface - JCR constants
4. UDRTextField - JCR customized text field
5. UDRComboBox - JCR customized combo box
7. RecordFilterPanel - JCR “Enter Record Filter” panel
8. QueryPanel - JCR “Query Console” panel
9. IntroPanel - JCR introduction panel
10. OutputSortOrderPanel - JCR “Select Output Fields and SortOrder” panel
11. GUIInterface - JCR resources constant as font, color, cursor, border
12. QueryManager - Handle main JCR logic as user creates a new query, user modifies a query, user deletes a query and user runs a query.

13. QueryManagerListener - Event listener for any changes to a query.

14. ImageManager - Load image and icon resources

15. IniBundle - Load JCR client configuration file

16. HttpCommManager - Manage different HTTP communication servers

17. HttpMessage - HTTP communication message

18. CommManager - Base class of different type of communication manager classes such as HTTPCommManager and RMICommManager

19. CommManagerGroup - Manage different kinds of communication manager classes as for RMI and HTTP

20. DisplayCardListener – Listen to the user mouse click event and display the appropriate JCR panel to the user.

**Server servlet classes:**


2. UsrDefnRepServerConfig - Load and read JCR server configuration file

3. FoxproDBMS - Handle database transactions as insert, update, delete and select, and it is designed specific for Microsoft Foxpro database.

**Serialized classes that are shared between client applet and server servlet:**

1. ClientServerMsg - Generic class to hold information that is passed between the JCR server servlet and the JCR client applet.
2. ERecord - Implement a database record
3. ERecordSet - Implement a database record set
4. SQLFilter - Implement a SQL filter, example “AND fieldname=fieldvalue”
5. SQLField - Base class of SQLFilter
6. QueryStruct - Implement a query

Configuration files:

UsrDefnRep.properties - JCR client applet configuration file
UsrDefnRepServerConfig.properties - JCR server servlet configuration file

Udr_querylist.dbf - Contain user created queries

Resource files:

downblack.gif
downblue.gif
downred.gif
print.gif
upblack.gif
upblue.gif
upred.gif
usrdefnreplogo.gif
edulogUsrDefnRepLogo.gif
5.2. **Class details.**

The JCR class details that list every methods and properties of each JCR class can be seen in Appendix C, D and E.

5.3. **JCR client and server design.**

5.3.1. **JCR client design**

The JCR client class diagram can be seen in Appendix F.

UsrDefnRepApplet implements the JCR client applet. UserDefnRepApplet extends javax.swing.JApplet to inherit all the attributes and functions of a standard applet and implements GUIInterface interface to be able to access GUI resources that are
customized for JCR as border style, color, font style, font size, cursor style and string constants.

UsrDefnRepApplet contains each of the following objects:

- An instance of java.net.URL and an instance of java.applet.AppletContext to allow the applet location to be located in the Internet. As the result, location of the JCR applet resources and configuration files can also be located.
- An instance of IniBundle to load the JCR applet configuration file.
- An instance of ImageManager to load the JCR applet icons, logos and image files.
- An instance of QueryManager to handle actions that user deletes a query, user updates a query update, user inserts a query query and user runs a query.
- An instance of CommManagerGroup to manage the communication with JCR server servlet, in this case, which is UsrDefnRepServer servlet.
- An instance of UsrDefnRepGUIManager to construct the JCR GUI and to handle user’s events such as user clicks on a button, user selects a field, and user enters a value into a text field.

QueryManager implements GUIInterface and java.ioRunnable interface. QueryManager contains an instance of java.util.Vector that holds the list of user queries and an instance of java.util.Hashtable that holds user query field ids and user query field descriptions.

CommManagerGroup could contain one or many instances of HTTPCommManager, each HTTPCommManager present communications with each server servlet. In JCR,
CommManagerGroup contains one instance of HTTPCommManager that presents the communication with UsrDefnRepServer servlet. HTTPCommManager extends CommManager abstract class that is the base class of all different communication classes for Http and Rmi communication methods. HTTPCommManager contains an instance of HttpMessage that implements http GET and POST request. HttpMessage contains an instance of java.net.URL that presents the internet location of the server that HttpMessage will send GET and POST request command to.

UsrDefnRepGUIManager contains one instance of javax.swing.JToolBar, three instances of javax.swing.JButton and five instances of javax.swing.JToggleButton that present the JCR right menu with the “Introduction”, “Query”, “Enter Record Filter”, “Output / Sort Order”, “Custom Report”, “Run Query”, “Standard Reports” and “Exit” buttons. UsrDefnRepGUIManager contains two instances of javax.swing.JLabel that presents the JCR logo and JCR version label. UsrDefnRepGUIManager contains four instances of java.awt.event.ActionListener that handle user events as user clicks on the JCR logo to go to Edulog Web Reports product page, user clicks on the “Run Query” button to execute a selected query, user clicks on the “Stand Reports” button to go to Web Standard Reports and user clicks on the “Exit” button and go to Edulog homepage. UsrDefnRepGUIManager contains five instances of DisplayCardListener that display the corresponding panel to user when user click on the “Introduction”, “Query”, “Enter Record Filter”, “Output / Sort Order” and “Custom Report” button.

UsrDefnRepGUIManager also contains each of following objects:
- An instance of IntroPanel that presents the JCR introduction panel.
- An instance of QueryPanel that presents the JCR “Query Console” panel.
- An instance of RecordFilterPanel that presents the JCR “Enter Record Filter” panel.
- An instance of OutputSortOrderPanel that presents the JCR “Select Output Fields and Sort Order” panel.
- An instance of ReportPanel that presents the JCR “Custom Report” panel.

IntroPanel, QueryPanel, RecordFilterPanel, OutputSortOrderPanel and ReportPanel all extend javax.swing.JPanel to inherit all attributes and functions of a standard panel container. They implement GUIInterface and/or UsrDeftiRepGUIInterface to be able to access JCR constants and GUI resources that are customized for JCR. QueryPanel, RecordFilterPanel, OutputSortOrderPanel and ReportPanel all also implement QueryManageListener interface and use an instance of QueryManager to manipulate user’s queries.

IntroPanel contains an instance of java.net.URL that present the internet location of JCR html introduction and help pages, an instance of javax.swing.JEditorPane that presents the editor that display JCR html introduction and help pages and an instance of javax.swing.event.HyperlinkListener that handle user event as user click on a hyperlink of JCR html page.
QueryPanel contains three instances of javax.swing.JLabel that present the “Query Console”, “Select desired query and choose the Run Query button:” and “Or enter the name of the new query and choose the Create New Query button” labels in the “Query Console” panel, one instance of javax.swing.JList that displays the user query list, one instance of javax.swing.JToolBar that present the tool bar and five instances of javax.swing.JButton that present the “Refresh Query”, “Copy Query”, “Rename Query”, “Delete Query” and “Create Query” button. QueryPanel also contains five instances of java.awt.event.ActionListener to handle user events as user clicks on the “Refresh Query” button, user clicks on the “Copy Query” button, user clicks on the “Rename Query” button, user clicks on the “Delete Query” button and user clicks on the “Create Query” button.

RecordFilterPanel contains one instance of UDRTextField that presents the “value” field in the “Enter Record Filter” panel, two instances of UDRComboBox that present the “field” and “comparison” drop down list, four instances of javax.swing.JLabel that present the “Enter Record Filter”, “Enter new filter and choose Add Filter button”, “Or select a filter and choose Delete Filter button” and “Filter List” labels, three instances of javax.swing.JRadioButton that present the “AND”, “OR” and “BLANK” radio buttons, two instances of javax.swing.JButton that present the “Add Filter” and “Delete Filter” buttons and one instance of javax.swing.JList that displays the list of existing filters of a query. RecordFilterPanel also contains two instances of java.awt.event.ActionListener to handle user events as user clicks on the “Add Filter” button and user clicks on the “Delete Filter” button.
OutputSortOrderPanel contains four instances of javax.swing.JLabel that present the “Select Output Fields and SortOrder”, “Select fields with your mouse and sort them with up/down arrow buttons”, “Select Output Fields” and “Select SortOrder” labels, four instances of javax.swing.JButton that present the “Save Output Fields & SortOrder”, “Restore Output Fields & SortOrder”, up arrow and down arrow buttons, two instances of javax.swing.JList that display the query available filter fields and the query selected filter fields, two instances of javax.swing.JRadioButton that present the “Ascending” and “Descending” radio buttons. OutputSortOrderPanel also contains one instance of ListSelectionListener that handles user event as user selects output fields for the query and four instances of java.awt.event.ActionListener that handle user events as user clicks on the “Save Output Fields & SortOrder” button, user clicks on the “Restore Output Fields & SortOrder” button, user clicks on the up arrow button and user clicks on the down arrow button.

ReportPanel implements java.awt.event.MouseAdapter to handle user event as user clicks on a query result record to display detail record in a new browser window. ReportPanel contains eight instances of javax.swing.JLabel that present the “Custom Report”, “Click on the column to sort data ascending/descending”, “Student Report”, “Number of Records”, “Student Record”, “Browser Print”, the number of record and the student record number label, one instance of javax.swing.JLabel that display the query result record set and one instance of javax.swing.JButton that presents the print button. ReportPanel also contain one instance of java.awt.event.MouseAdapter to handle user
event as user click on the column of the query result table to sort the data ascending or descending.

5.3.2. **JCR server design**

The JCR server class diagram can be seen in Appendix G.


UsrDefhRepServer contains an instance of UsrDefnRepServerConfig that extends java.util.PropertyResourceBundle to be able to read JCR server configuration file as resource stream. UsrDefhRepServer also contains an instance of FoxproDBMS to connect to database and perform SQL insert, update, select and delete transaction. In order to process requests from JCR client applet, UsrDefhRepServer contains many instances of java.io.ObjectInputStream and java.io.ObjectOutputStream and uses many instances of java.lang.reflect.Method. Everytime, JCR client applet sends a request to JCR server servlet, JCR server servlet creates an instance of java.io.ObjectInputStream (as seen from ‘request*’ in the server class diagram) to wrap around the request and analyze the request, and then the server servlet uses an instance of java.lang.reflect.Method to invoke the appropriate servlet service method. The result of
called service method will be wrapped in a created instance of java.io.ObjectOutputStream (as seen from ‘response*’ in the server class diagram) and sent back to JCR client applet.

FoxproDBMS uses an instance of sun.jdbc.odbc.JdbcOdbcDriver to load java database dynamic link library (dll) in to operating system memory and from that FoxproDBMS uses an instance of java.sql.Connection to create the database connection for a user-configured database server name (DSN). FoxproDBMS contains an instance of java.io.PrintWriter to log any database transactions for debugging purpose and also an instance of java.util.Hashtable that presents the database connection pool.

ClientServerMsg is the generic communication package of JCR. ClientServerMsg is used by both JCR client applet to send request and JCR server servlet to send response. JCR client applet sends an instance of ClientServerMsg include requested method name and parameters to the JCR server servlet. JCR server servlet examines the ClientServerMsg that is sent from JCR client applet to find out which service method the client applet wants to execute and what input parameter it want to use. The result from server servlet service method execution is embedded into an instance of ClientServerMsg and then sends back to JCR client applet. The JCR client applet finally examines the returned ClientServerMsg object, extract appropriate data and display them to user.

ClientServerMsg generic communication object uses ERecordSet, ERecord, SQLFilter, SQLField, QueryStruct, java.util.Vector, java.lang.String, java.lang.Boolean,
java.util.Hashtable object for suitable case of JCR client applet request and JCR server servlet response. All of these objects implement java.io.Serializable interface so that they can be tunneled through http communication.

5.4. **Description of the initialization process of the software.**

When a user first requests a JCR client applet from the server, an instance of the UsrDefnRepApplet class is created and is loaded into Java Virtual Machine (JVM). The UsrDefnRepApplet class then creates an instance of the UsrDefnRepGUIManager class, which presents the main GUI of JCR.

![UML Diagram]

UsrDefnRepGUIManager class creates the appropriate GUI objects by instantiating the IntroPanel class for the introduction panel, instantiating QueryPanel class for the “Query Console” panel, instantiating RecordFilterPanel class for the “Enter Record Filter” panel, instantiating OutputSortOrderPanel class for the “Output Fields and SortOrder” panel, instantiating the ReportPanel class for the “Custom Report” panel.
When the JCR client applet is initialized, UsrDefnRepApplet class creates an instance of the IniBundle class which reads JCR client configuration file "UsrDefnRepConfig.properties" so that the client applet knows where to load icon, image, HTML file resources and where the JCR server servlet is located. At this time, the DefnRepApplet class also creates an object of the ImageManager class. With information is read from the "UsrDefnRepConfig.properties" configuration file, the ImageManager object loads the appropriate icons, images and HTML files for JCR client applet.
When UsrDefnRepApplet is instantiated, UsrDefnRepApplet requests the JCR UsrDefnRepServer servlet for the list of existing queries that are in the database and displays the query list on the “Query Console” panel. JCR UsrDefnRepServer servlet is initialized after JCR UsrDefnRepApplet invokes it. When UsrDefnRepServer class object is created, UsrDefnRepServer class also creates an instance of UsrDefnRepServerConfig class that reads the “UsrDefnRepServerConfig.properties” configuration file and an instance of FoxproDBMS class that handles database insert, update, delete and select transaction.
requests*: JCR client applet requests JCR server servlet for the existing user queries in the database.

returns*: JCR server servlet returns the query list back to JCR client applet.

5.5. **Software document**

The JCR software document that describes all JCR class methods in details can be seen in Appendix H.

6. **Code Development**

6.1. **Choice of programming tools and environment**

There are many different programming tools for Java, most well-known are JBuilder from Borland, Visual Café from Symantec, Visual Age from IBM, Microsoft Visual J from Microsoft.
The Java group at Edulog has not agreed on an integrated development environment (IDE). The reason is that different people before joining Java programming group had been using different editing tools for programming as Code Warrior, Microsoft Visual Studio, JCreator (a freeware editor), Vim (an improved version of vi) and they are very used to the developing tools that they have been using.

I have been using JBuilder 5.0 and Vim in my programming. While JBuilder 5.0 is an expensive commercial programming tool, Vim is free software. Though JBuilder 5.0 has more significant features than Vim, JBuilder 5.0 requires a lot of RAM and hard disk space. Therefore, JBuilder 5.0 takes quite some time to load. In addition, JBuilder 5.0 is more as a mouse-click tool: it means that user uses mouse to activate commands rather than keyboard. I personally prefer to activate commands with keyboard to mouse-click since it is much faster in editing. So Vim was chosen as the development tool in JCR project.

6.2. **Coding standard.**

Since JCR is developed for Edulog, the code must be consistent with Edulog coding standard. Using this coding standard in implementing means that JCR source code will be easier to maintain as well as easier to integrate with other Edulog applications.
The coding standard includes rules about class names, method names, variable names, index names, spacing and packages. Below are some important features of Edulog coding standard and their advantages:

- Variable names use Hungarian names, which is popularly used in Microsoft Visual C and C++. This is an advantage for VC and C++ programmers who change to Java programmers as they have seen similar features. The Hungarian name convention also allows developers to tell what type of variable is without looking up the declaration codes. Example: for a double variable which presents length of a trip, it is named a “dLength”, by looking at its name the initial letter “d” tells the developer that “dLength” is a double type variable.

- Index uses 2 spaces for any new line indexing. This rule is to avoid developer to use “tab” when he or she goes to a new line. Since a tab is set by a value of different number of spaces in different IDEs (for example: tab in Jbuilder 4.0 by default is set to 6 spaces, tab in Vim by default is set to 4 spaces and tab in Microsoft Visual J by default is set to 8 spaces), when a developer edits a Java file created by a different IDE, the Java file will not align correctly.

- Variable declaration should not be in a while or for loop. This avoids duplication of declaration of variables. Example: “for (int nIndex = 0; nIndex < 100; nIndex++)” should be rewritten as “int nIndex; for (nIndex = 0; nIndex < 100; nIndex++)”.

The Edulog coding standard for Java context can be seen from Appendix I.
6.3. **Program resource management.**

JCR resource management includes html files, images, icons, JCR client applet configuration file and JCR server servlet configuration file management.

JCR client applet configuration file consists of information of where to load icons and logos for the JCR applet, which server servlet the JCR applet connects to and which URL the JCR html files are located.

JCR server servlet configuration file consists of information of where the transaction information is logged for debug, which database is used and what the limitation of number of records per query is.

All the images, icons, logos used in JCR and JCR applet classes will be packed in an client applet jar file. The jar file is a type of compression file with compressed ration of 0 %. The advantage of this is that instead of many times browser sends request to load each image, each icon, each logo or each Java class, the browser only needs to send the request once to load all the resources for JCR applet.
7. Testing

7.1. **Class Testing.**

Class testing includes instantiating class objects, passing parameter between classes and accessing class members. In each class, at implementing and testing phase a main method “public static void main(String[] args){}” is included to examine the instantiating class object and the accessing class members to see if it behaves as desired. The main method used for testing purpose in each class is basically as follows:

```java
public static void main(String[] args)
{
    try
    {
        static Class Name class = new Class Name(arg1,arg2,arg3);
        System.out.println("Instantiation succeed");
        System.out.println("Access member variable A: " + class.A);
        System.out.println("Access method M(): " + class.M(arg1,arg2));
    }
    catch(Exception e)
    {
        e.printStackTrace(System.out);
    }
}
```
In order to test a single JCR class, from command line user types: "Java ClassName".

Test driver is also used to test parameter passing between classes. The test driver is a separate class which instantiated several JCR classes and output parameters passing between these class instances to system output (system output is often user computer screen). A simple typical example could be as follows:

```java
public class TestDriver1 {

    public TestDriver1() {

    }

    public static void main(String[] args) {

        try {

            static JCRObject1 object1 = new JCRObject1(org1);
            static JCRObject2 object2 = new JCRObject2(arg2);

            System.out.println("Testing on pass parameters from object1 to a method of object2");
            System.out.println(object2.M(object1.A));

        } catch(Exception e) {

            e.printStackTrace(System.out);

        }

    }

}
```
7.2. **Functional testing.**

Functional testing includes a series of test sequences to see if JCR functions correctly when performing its required features. In order to implement this, a more complicated “test driver” is written. A “test driver” for this often consists of instantiating JCR class objects, constructs GUIs and output proper result for specific input. For example, to examine the help html file is display correct in the introduction panel, a IntroPanel class is instantiated, and method “run()” is called with proper “url” that points to html help file is set in configuration file.

- The configuration file is modified to add “INTROPAGE” property that points to “IntroPage.html” for testing.

```plaintext
# File: UsrDefnRepConfig.properties
# Version: v0.0.1
# Modification: 08/26/2001 10:14 PM

# LOGO
USRDEFNREP_LOGO=usrdefnreplogo.gif

# ICONS
DOWNBLACKICON=downblack.gif
DOWNBLUEICON=downblue.gif
DOWNREDCON=downred.gif
```
- A simple test driver to test if the "IntroPage.html" help file is displayed correctly in the introduction panel:

```java
Class TestHelpFileDisplay {
    private static URL m_introURL;
    public static void main(String[] args)
    {
        try
        {
            m_introURL = new URL(UsrDefnRepApplet.getIniBundle().getString("SERVERS", "INTROPAGE"));
            static IntroPanel intropanel = new IntroPanel();
            intropanel.run(m_introURL);
        }
        catch(Exception e)
        {
            e.printStackTrace(System.out);
        }
    }
}
```
7.3. **Edulog testing procedure.**

When programmer group completes a version, the group labels the version of the source codes and the version of the compiled application. The application is then sent to testing group. The testing group tests the application in two steps:

1. Bugs fixes and enhancement features testing

2. Regression testing.

Bugs fixes testing is to test all the fixes that are made to bugs, which were found in the previous version. Enhancement feature testing is to test the new enhancement features.

Regression testing is to test all the functional features of the application and to go through all the situations in which the user might use the application to see if the application exhibits failures or crashes.

All the bugs found are logged into a web bug database, where the programmer can access within or outside the company. There are 5 bug categories:

- Priority-1 bugs: these bugs are those that crash the application. Priority 1 means very high priority and requires to be fixed as soon as possible.
- **Priority-2 bugs**: these bugs are those that cause serious errors and happen often when user uses the application.

- **Priority-3 bugs**: these bugs are those that cause less serious errors when user uses the application.

- **Priority-4 bugs**: these bugs are those that cause insignificant errors when user uses the application.

- **Priority-5 bugs**: these bugs are those that cause unnoticeable and insignificant errors and they rarely occur when user uses the application.

All the bugs of priority from 1 to 3 are considered as high priority and required to fix before the next version release.

8. **Conclusion.**

The JCR software first version was completed in July 2001. Together with the Edulog Web Standard Reports software, the JCR makes a complete web reporting software package. This web reporting software package has been deployed in quite a number of school districts. The largest school district that has the software package is the school district of more than 200,000 students in California. Two oversea school districts in Toronto, Canada and in Paris, France also use the software package for their reporting.
The JCR has provided users with flexibility and convenience in creating a report through the network access to a database. It is easy to set up and flexible in configuration. The current version JCR allows the user to configure the JCR to query report for only a single table in a database and does not have the capability to handle the SQL join condition from multiple tables. A future enhancement of the JCR could be to implement a feature, which allows the JCR to generate query reports from multiple database tables.


1. Requirements.
   1.1. Must allow the user to select criteria by which to filter students in a query.
   1.2. Must allow those queries to be saved, changed, deleted, copied, and renamed.
   1.3. Must create a report of students returned by a query.
   1.4. Must allow the user to select output fields and choose sort order.
   1.5. Once a query has been selected, it must be able to transfer that information into a Select Data request and display only those students returned from the query.
   1.6. Must provide instructions to guide user how to use the application.
   1.7. Must display version of the application.


User-Defined Reports

User-Defined Reports allows you to search through the student database to find students matching certain criteria. It is possible to search on any criteria in the database and to save these searches as queries enabling you to perform the same report as needed. Further, selecting any of the students returned by a query will bring up a page with all the relevant information for that student.

Using this tool is quite simple. First, select one of the current queries from the Query Console (accessed through the Query button to the right), or create a new query by typing a name for the query and selecting the Create Query button. Next proceed to the Enter Record Field window where you can then select various fields in the database and set comparators for them such as: Student Last Name like 'Smith'. Next, you can proceed to the Output/Sort Order window to select the information you wish to have included in the report. Highlighted fields will be included in the report. Use the arrow keys to change the Sort Order. Once everything is to your liking, select the Run Query button and a report will be generated and will return information that matches your search criteria.

For more detailed information concerning each individual window of this module, please refer to the links below:

- The Query Console
- Enter Filter Records

2.1. Tab Order:
   2.1.1. Introduction>> Query>> Enter Record Filters>> Output/Sort Order>> Custom Reports>> Run Query>> Standard Reports>> Exit
2.2. Window Behavior:
2.2.1. Always opens to the Introduction Page.
2.2.2. Exit takes the user to www.edulog.com
2.2.3. The Edulog label in the top right hand corner will also take the user to www.edulog.com

2.3. Components:
2.3.1. Introduction: Button, on Mouse Click, or in focus and enter, opens the Introduction Page (default window).
2.3.2. Introduction Page: HTML, provides an overview of how to use the system with links to more detailed information for the various parts.
2.3.3. Query: Button, on Mouse Click, or in focus and enter, opens the Query Console window.
2.3.4. Enter Record Filters: Button, on Mouse Click, or in focus and enter, opens the Enter Record Filters window.
2.3.5. Output/Sort Order: Button, on Mouse Click, or in focus and enter, opens the Output and Sort Order window.
2.3.6. Custom Report: Button, on Mouse Click, or in focus and enter, opens the Custom Report window where reports are displayed.
2.3.7. Run Query: Button, on Mouse Click, or in focus and enter, fires the Run Query action.

3. Query Console window.

![Query Console Image]
3.1. Tab Order:
   3.1.1. Text Area>> Refresh Query>> Copy Query>> Rename Query>> Delete Query>> Text Field>> Create Query.

3.2. Components:
   3.2.1. List: List, Lists all the queries that have been created and saved. Selecting a query fires the Selected Query action. Does not allow multiple selection.
   3.2.2. Refresh Query: Button, on Mouse Click, or in focus and enter, fires the Update Query List action.
   3.2.3. Copy Query: Button, on Mouse Click, or in focus and enter, fires the Copy Query action.
   3.2.4. Rename Query: Button, on Mouse Click, or in focus and enter, fires the Rename Query action.
   3.2.5. Delete Query: Button, on Mouse Click, or in focus and enter, fires the Delete Query action.
   3.2.6. Text Field: Text Field, provides for the Rename Query and Create Query actions.
   3.2.7. Create Query: Button, on Mouse Click, or in focus and enter, fires the Create Query action.

4. Enter Record Filters window.

![Enter Record Filter Window]

- AND
- OR
- BLANK

**Field**

**Comparison**

**Value**

- Grade

**Filter List**

- School = 400
- AND Grade = 00
- AND Grade = 12
- AND Grade = 10

**Buttons**

- Add Filter
- Delete Filter

Applet started.
4.1. Tab Order:
   4.1.1. AND » OR » BLANK » Field » Comparison » Value » Add Filter » Filter List » Delete Filter

4.2. Components:
   4.2.1. **AND**: Radio Button, When selected causes new filters to use the AND operator in conjunction with other filters.
   4.2.2. **OR**: Radio Button, When selected causes new filters to use the OR operator in conjunction with other filters.
   4.2.3. **BLANK**: Radio Button, used only for the first filter in a list of filters. Has no meaning otherwise.
   4.2.4. **Field**: Combo-box, lists all the fields in the Student.dbf file. One must select a field, as well as a comparison type and a comparison value in order to create a valid filter.
   4.2.5. **Comparison**: Combo-box, lists all the possible comparisons. Defaults to empty.
   4.2.6. **Value**: Text Field, holds the value that to be compared to the field.
   4.2.7. **Add Filter**: Button, on Mouse Click, or in focus and enter, fires the Add/Update Filter action.
   4.2.8. **Filter List**: List, lists all the filters for this query. Double Mouse Click on a filter will enter that filter's relevant information into the matching fields above and prepare that filter for update.
   4.2.9. **Delete Files**: Button, if a filter is selected, on Mouse Click, or in focus and enter, fires the Delete Filter action.

5. **Select Output and Sort Order window.**
5.1. Tab Order:
5.2. Select Output Fields>> Select Sort Order>> Asc/Desc Sort
Ascending>> Asc/Desc Sort Descending>> Save Output Fields & Sort
Order>> Restore Output Fields & Sort Order>> Up Arrow>> Down
Arrow

5.2. Components:
5.2.1. Select Output Fields: List, displays all the fields from the Studnt.dbf
file. Highlighted fields are selected and immediately shown in the Select
Sort Order List. Allows multiple selection.
5.2.2. Select Sort Order: List, displays all the currently selected Output
fields. Allows only single selection. Selecting a row and then Mouse
Click on one of the Arrow buttons fires the Increase Sort Order Priority
action or the Decrease Sort Order Priority action (depending on the arrow
used.).
5.2.3. Ascending: Radio Button, marks the report as sorting in ascending
order, defaults to selected.
5.2.4. Descending: Radio Button, marks the report as sorting in
descending order, defaults to not selected.
5.2.5. Save Output Fields & Sort Order: Button, on Mouse Click, or in
focus and enter, fires the Save Output/Sort Order action.
5.2.6. Restore Output Fields & Sort Order: Button, on Mouse Click, or in
focus and enter, fires the Restore Output/Sort Order action.
5.2.7. Up Arrow: Button, if a row in Select Sort Order is highlighted, on
Mouse Click, or in focus and enter, fires the Increase Sort Priority
action.
5.2.8. Down Arrow: Button, if a row in Select Sort Order is highlighted,
on Mouse Click, or in focus and enter, fires the Decrease Sort Priority
action.

### 6.1. Components:

6.1.1. **Student Report**: Table, lists all the students returned by a query. Defaults to empty with ten columns.

6.1.2. **Column Headers**: Columns, on Mouse Click, fires the **Column Sort** action.

6.1.3. **Rows**: Mouse Click, or in focus and enter, on row fires the **Student Information** action.

6.1.4. **Number of Records**: Lists total number of records returned.

6.1.5. **Selected Record**: Displays the last record selected, in a numeric form counting with one as the first record (top-most record) and all other records incremented from that.

### 7. Actions.

7.1. **Run Query**

7.1.1. Executes the currently selected query, including any changes, whether they've been saved or not.

7.1.2. If no records are returned, displays an error dialog stating so.

7.1.2.1. Message: No students matched your criteria. Please try again with broader filters.

7.1.2.2. OK, closes dialog, no further action.
7.1.3. If more than the set number of records are returned (the total number of records that may be returned is set in the properties file) an error dialog displays.

7.1.3.1. Message: More than (max returned records allowed) students were returned by your search. Please refine your filters and try again.
7.1.3.2. Ok, closes dialog, no further action.

7.1.4. Changes the current view to the Custom Report window.
7.1.5. Fills the table in the Custom Report window with the student records returned by the search.
7.1.6. Changes the cursor while processing.

7.2. Selected Query:
7.2.1. Occurs whenever a different query is selected.
7.2.2. The simple query is the default query and is automatically selected at load time.
7.2.3. Fills the Enter Record Filters with the filters for the query.
7.2.4. Fills the Output Fields and Sort Order window with the appropriate values for this query.
7.2.5. When Run Query is activated, this is the query that is run.

7.3. Update Query List:
7.3.1. Checks the server for new queries.
7.3.2. Updates the query list with any and all new queries.

7.4. Copy Query:
7.4.1. Creates a new query in the Query list with all the same attributes of the selected query.

7.5. Rename Query:
7.5.1. Checks for a name in the text field of the Query Console. If no name, displays an error message asking the user to enter a name into the text field. No further action.
7.5.2. Checks if the query selected is the Simple Query. If yes, then no further action.
7.5.3. Renames the selected query to the name written in the text field.

7.6. Delete Query:
7.6.1. Checks if the query selected is the Simple Query. If yes, then no further action.
7.6.2. Otherwise, Deletes the query from the server and refreshes the list.

7.7. Create Query:
7.7.1. Checks for a name in the text field of the Query Console. If no name, displays an error message asking the user to enter a name into the text field. No further action.
7.7.2. Creates a query with the new name on the server. Query defaults to no filters and all the Output Fields selected. Server is updated, and the query list is refreshed.

7.8. Add/Update Filter action:
7.8.1. Checks whether the filter is a new filter, or an existing filter to be updated.
7.8.2. If a new filter, then it adds the filter to the list.
7.8.3. If it is a current filter being updated, then it updates the selected filter.

7.9. Delete Filter:
7.9.1. Checks that a filter is selected. If yes, then it deletes that filter. If no, then performs no action.

7.10. Increase Sort Priority:
7.10.1. Checks that a row in the Select Sort Order list is highlighted. If no, no further action.
7.10.2. Moves the selected row up one level, displacing the item above and setting that item below the selected row.
7.10.3. If the highlighted row is at the top of the list, no further action.

7.11. Decrease Sort Priority:
7.11.1. Checks that a row in the Select Sort Order list is highlighted. If no, no further action.
7.11.2. Moves the selected row down one level, displacing the item below and setting that item above the selected row.
7.11.3. If the highlighted row is at the bottom of the list, no further action.

7.12. Save Output/Sort Order:
7.12.1. Saves the current Output and Sort Order settings to the query.

7.13. Restore Output/Sort Order:
7.13.1. Replaces currently selected Output Fields and Sort Order with the last saved version. Any changes since the last Save Output/Sort Order action are discarded.

7.14. Column Sort:
7.14.1. Checks the columns current sort order, if no sort order, it then sorts the column in ascending order. If the column is in ascending order, it sorts the column in descending order. If the column is in descending order, it sorts the column in ascending order.

7.15. Student Information:
7.15.1. Opens a new browser window.
7.15.2. Fills the new window with all the information for the selected student.
Appendix B: Associated sequence diagram for the use case where the user creates a query, defines filters for the query, defines output fields and sort order for the query and finally runs the query.

User selects "Query" button to go to "Query Console" panel

DisplayCardListener: actionPerformed() \rightarrow ::UsrDefnRepGUIManager

After enter query name, user selects "Create Query" button to create the query

QueryManager:createQuery() \rightarrow ::UsrDefnRepServer

UsrDefnRepServer: insert_UDR_Query() \rightarrow Edulog Database

User selects "Enter Record Filter" button to go "Enter Record Filter" panel

DisplayCardListener: actionPerformed() \rightarrow ::UsrDefnRepGUIManager

\rightarrow ::RecordFilterPanel
After user selects a filter field, a filter operator Operator and enter a value to filter value field, user selects “Add Filter” button

```
QueryManager: addFilter()
```

```
UsrDefnRepServer: update_UDR_Query()
```

User selects “Output/Sort Order” button to go to “Select Output Fields and SortOrder” panel

```
DisplayCardListener: actionPerformed()
```

After user selects output fields from Output field and sort order, user selects “Save Sort Order & Sort Order” button

```
QueryManager: saveFieldList()
```

```
UsrDefnRepServer: update_UDR_Query()
```

Edulog Database

```
::UsrDefnRepGUIManager
```

Edulog Database

```
::RecordFilterPanel
```

```
::UsrDefnRepServer
```

```
::OutputSortOrderPanel
```

```
::UsrDefnRepGUIManager
```

```
::OutputSortOrderPanel
```

```
::UsrDefnRepServer
```

```
::OutputSortOrderPanel
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```

```
::UsrDefnRepServer
```
User selects “Run Query” button to run the Query

::UsrDefnRepGUIManager

QueryManager: generateReport()

::UsrDefnRepServer

UsrDefnRepServer: run_UDR_Query()

Edulog Database

The query result set is returned from the sever and is displayed on “Custom Report” panel for user to view

ReportPanel: viewReport()
Appendix C: **JCR client applet classes.**

1. **UserDefnRepApplet**

```java
UserDefnRepApplet extends javax.swing.JApplet
    implements UsrDefnRepInterface

private static AppletContext m_contextApplet
private URL m_urlAppletCodebase
private QueryManager m_querymanager
private UsrDefnRepApplet m_muserdefnrepapplet
private String m_strReportServerUrl
private String m_strIntroPageUrl
private static String m_strVersion
private static ImageManager m_imagemanager
private static CommManagerGroup m_commManagerGroup
private static IniBundle m_inibundle

public void init()
public static ImageManager getImageManager()
public static IniBundle getIniBundle()
public static CommManagerGroup getCommManagerGroup()
public static AppletContext getEnvironment()
public static String getVersion()
```

2. **UsrDefnRepInterface**

```java
interface UsrDefnRepInterface

public static char COMMA_REPLACEMENT_DELIMETER
public static int MAX_NUMER_FILTERS
public static String CUSTOMREPORT
public static String SUBREPORTFILTER
```

3. **UsrDefnRepGUIManager**

```java
UserDefnRepGUIManager extends javax.swing.JPanel
    implements GUIInterface

UserDefnRepGUIManager(QueryManager,URL,URL)
public JPanel getPanelDesktop()
public JToggleButton getReportButton()
private void BuildButtonBar(URL,URL)
private JToolBar buildButtonBar(URL,URL)
private JPanel buildDesktopPanel()
private AbstractButton buttonProps(AbstractButton,Color)
```
4. IntroPanel

```java
IntroPanel extends java.swing.JPanel
    implements GUIInterface, UsrDefnRepInterface

private URL m_urlIntro
private JeditorPane m_editorpanelIntro

IntroPanel()
private void buildIntroPanelGUI()
public void run()
```

5. QueryPanel

```java
QueryPanel extends javax.swing.JPanel
    implements QueryManagerLister, GUIInterface

private QueryPanel m_querypanel
private Jlist m_listQuery
private JTextField m_textfieldQueryName
private QueryManager m_querymanager
private JButton m_buttonRenameQuery
private JButton m_buttonDeleteQuery

QueryPanel(QueryManager)
public QueryPanel getInstance()
public runQueryInfo(int)
private buildQueryPanelGUI()
private Jlabel buildQueryPanelTitle()
private buildQueryListPanel()
private JPanel buildCreateQueryPanel()
private DefaultListModel buildListModel(Object[])
private AbstractButton buttonProps(AbstractButton,Color)
```
6. OutputSortOrderPanel

OutputSortOrderPanel extends javax.swing.JPanel
implements QueryManagerListener, GUIInterface

private OutputSortOrderPanel m_outputsortorderpanel
private Boolean m_bAscending
private Boolean m_bDescending
private int[] m_arrayPreviousIndices
private JButton m_btnSaveOutputField
private JButton m_btnRestoreOutputField
private JButton m_btnSortUp
private JButton m_btnSortDown
private JList m_listOutputField
private JList m_listSortOrder
private JRadioButton m_rbuttonDESC
private QueryManager m_querymanager

OutputSortOrderPanel(QueryManager)
public OutputSortOrderPanel getInstance()
public void reviewQueryInfo(int)

7. ReportPanel

ReportPanel extends javax.swing.JPanel
implements QueryManagerListener, GUIInterface

private ReportPanel m_reportPanel
private JTable m_tableCustomReport
private ERecordSet m_recordset;
private int m_nColumnIndex
private boolean[] m_arraySORT_A_TO_Z
private JLabel m_labelNumberRecords;
private JLabel m_labelSelectedRecords
private QueryManager m_querymanager
private JTableHeader m_tableheader
private Hashtable m_hashTableHeaderDesc

ReportPanel(QueryManager)
public ReportPanel getInstance()
public void sortByColumn(int)
public void viewReport(ERecordSet)
public void displayReport(Object[],Object[])
public void mouseClicked(MouseEvent me)
public void mouseEntered(MouseEvent me)
public void mouseExited(MouseEvent me)
public void mousePressed(MouseEvent me)
private JLabel buildReportPanelTitle()
private JPanel buildReportTablePanel()
private String removeDotNotation(String)
8. RecordFilterPanel

RecordFilterPanel extends javax.swing.JPanel
implements QueryManagerListener, GUIInterface, UsrDefnRepInterface

private RecordFilterPanel m_recordFilterPanel
private QueryManager m_querymanager
private JList m_listFilters;
private JRadioButton m_rbuttonAND
private JRadioButton m_rbuttonOR
private JRadioButton m_rbuttonBLANK
private UDRComboBox m_comboboxSelectField
private UDRComboBox m_comboboxSelectComparison
private UDRTextField m_textfieldEnterValueField
private int m_selectedFilterIndex
private JButton m_btnDelete
private JButton m_btnAdd

RecordFilterPanel(QueryManager)
public void reviewQueryInfo(int)
private void BuildRecordFilterPanelGUI()
private JLabel buildRecordFilterPanelTitle()
private JPanel buildFilterFieldPanel()
private JPanel buildFilterListPanel()
private DefaultListModel buildListModel(Object[])
private AbstractButton buttonProps(AbstractButton,Color)
private String removeComas(String)

9. QueryManagerListener

interface QueryManagerListener

public void reviewQueryInfo(int)

10. DisplayCardListener

DisplayCardListener implements java.awt.event.ActionListener

protected Container m_container
protected String m_cardName

public DisplayCardListener(String,String)
public void actionPerformed(ActionEvent)
11. QueryManager

```
QueryManager implements java.lang.Runnable, GUIInterface

private Hashtable m_hashFieldName
private Hashtable m_hashFieldDesc
private Vector m_vectorQuerySet
private Vector m_vectorQueryManListener
private QueryStruct m_queryThread
private HTTPCommManager m_httpComm
private int m_nQueryRecSetId

QueryManager()
public void run()
public void printFieldDescriptionNameHashtables()
public Object[] getFieldDescriptionSet()
public String getFieldDescriptionFor(String)
public Object[] getFieldNameSet()
public String getFieldNameFor(String)
public int getNumberOfQueries()
public Object[] getQueryDescriptions()
public String copyQuery(int, String)
public void createQuery(QueryStruct)
public void deleteQuery(int)
public void renameQuery(int, String)
public void selectQuery(int)
public void selectQueries()
public String printFields()
public Object[] restoreFieldList()
public void saveFieldList(Object[])
public void addFilter(String, String, String, String)
public void deleteFilter(int)
public Object[] getFilterList()
public int getNumberOfFilters()
public String getFilterCondition(int)
public String getFilterField(int)
public String getFilterComparison(int)
public String getFilterValue(int)
public void modifyFilter(int, String, String, String, String)
public String getFieldSortOrder()
public void setFieldSortOrder(String)
public void printFilters();
public QueryManagerListener addListener(QueryManagerListener)
public void removeListener(QueryManagerListener)
public void notifyListeners()
public void generateReport()
public void showStudentReport(int)
public void showQueryReport()
private void loadFieldSet()
private void loadQuerySetQ
```
12. UDRComboBox

UDRComboBox extends JPanel
    implements GUIInterface

private JComboBox m_comboboxSelection
private JLabel m_label

UDRComboBox()
    public void setEnabled(Boolean)
    public String getSelectedItem()
    public void setSelectedItem(String)
    public void setOpaque(Boolean)

13. UDRTextField

UDRTextField extends JPanel
    implements GUIInterface

private JTextField m_textfield
private JLabel m_label

UDRTextField(String)
UDRTextField(String, Dimension)
    public String getText()
    public void setText(String)
    public void setEnabled(boolean)
    public void setOpaque(boolean)

14. CommManagerGroup

CommManagerGroup

public Hashtable serverGroup

CommManagerGroup()
    public void addCommManager(CommManager, String)
    public HTTPCommManager createHTTPCommManager(URL)
    public HTTPCommManager createHTTPCommManager(URL, AppletContext)
    public RMICCommManager createRMICCommManager(String, String)
    public RMICCommManager createRMICCommManager(String, String, String)
    public CommManager getCommManager()
15. HTTPCommanager

HTTPCommanager extends CommManager

```java
private AppletContext env
private HttpMessage httpMsg
private URL server
```

HTTPCommanager(String)
HTTPComManager(String,AppletContext)
HTTPCommManager(URL)
HTTPCommManager(URLAppletContext)

public InputStream sendServerGETRequest APPLET(Properties)
public InputStream sendServerGETRequest APPLET(String)
public InputStream sendServerGETRequest BROWSER(Properties)
public InputStream sendServerGETRequestBROWSER(String)
public Object sendServerPOSTRequest(String,Object[])
public Object sendServerRequest(String,Object[])
public boolean testCommLink()

16. CommManager

abstract CommManager

```java
CommManager()
public abstract Object sendServerRequest(String,Object[])
public abstract boolean testCommLink()
```

17. HTTPMessage

HttpMessage

```java
private URL servlet
```

HttpMessage(String)
HttpMessage(URL)

public InputStream sendGetMessage(Properties)
public InputStream sendGetMessage(String)
public InputStream sendPostMessage(Serializable)
public String toEncodedString(Properties)
18. GUIInterface

```java
interface GUIInterface {
    Border loweredBevelBorder
    Border raisedBevelBorder
    Border emptyBorder5
    Border emptyBorder50
    String INTRODUCTION
    String QUERY
    String FILTER
    String OUTPUTSORTORDER
    String REPORT
    Color DESKTOPGRAY
    Color WINDOWBACKGROUN
    Color MAROON
    Color DARKPINK
    Color PINK
    Color DARKGREEN
    Color LIGHTGREEN
    Color DARKBLUE
    Color PURPLE
    Color LIGHTYELLOW
    Font boldTimesRoman10
    Font boldTimesRoman11
    Font boldTimesRoman12
    Font boldTimesRoman14
    Font boldTimesRoman24
    Font normalTimesRoman9
    Cursor defaultCursor
    Cursor waitCursor
    Cursor handCursor
}
```

19. ImageManager

```java
ImageManager {
    private Hashtable m_hash
    ImageManager()
    public void add(String, URL)
    public Image get(String)
}
```
20. IniBundle

```java
package IniBundle;

import java.util.Hashtable;

public class IniBundle {
    private Hashtable m_hashGroups;
    private Hashtable m_hash;

    public IniBundle(URL url) {
        // Constructor implementation
    }

    public Enumeration getKeys(String key) {
        // Method implementation
    }

    public String getString(String key) {
        // Method implementation
    }

    public String getString(String key, String defValue) {
        // Method implementation
    }
}
```
Appendix D: **JCR server servlet classes.**

1. **UsrDefnRepServer**

   ```java
   UsrDefnRepServer extends HttpServlet
   
   private String CONTENT_TYPE
   private String CUSTOMREPORT
   private PrintWriter m_logFile
   private UsrDefnRepServerConfig m_configFile
   private foxproDBMS m_foxDBMS
   private String m_strEdulogmqMTablesCon
   private String m_strEdulogntEltTablesCon
   private String m_strUDRQueryList
   private String m_strCustomReportTable
   private int m_nMaxNumRecords
   private Vector m_vectorField_NameDesc
   private ERecordSet m_recordsetQuery
   private Hashtable m_hashFieldName
   private Hashtable m_hashFieldDesc
   private Hashtable m_hashQueryRecSet
   private int m_nQueryRecSetId
   
   public void init(ServletConfig)
   public void doGet(HttpServletRequest,HttpServletResponse)
   public void doPost(HttpServletRequest, HttpServletResponse)
   public void destroy()
   public Vector queryStudentReportFieldDesc()
   public Vector select_UDR_Queries()
   public String insert_UDR_Query(String,String,String)
   public Boolean delete_UDR_Query(String)
   public Boolean update_UDR_Query(String,String,String, String)
   public Hashtable run_UDR_Query(QueryStruct)
   private int getNumberOfRecordsQuery(QueryStruct)
   private String buildRunReportSQL(QueryStruct)
   private ERecord getStudentlnfor(String)
   private ERecordSet getStudentlnfor(String)
   private void printStudentReport(PrintWriter,String)
   private void printQueryReport(Integer ,String ,PrintWriter)
   private void printError(PrintWriter)
   private void handleUpdate(String ,String,String)
   private void handle250Update(String,String ,String )
   ```
2. FoxproDBMS

```java
public static final String ALL
public static final String DISTINCT
private Hashtable connectionPool
private Connection dbConn
private String DSN
private Hashtable fieldTypeHash
private static final int FOXPRO_MEMOFIEL_CHAR_LIMIT
private static final String FOXPRO_SQL_NEWLINE_SEPARATOR
private PrintWriter logfile
private logtransaction

public FoxproDBMS()
public FoxproDBMS(boolean, PrintWriter, Object[])
public FoxproDBMS(PrintWriter, boolean, String)
public void closeAllConnection()
public void closeConnection(String)
public void connectToDataSource(String)
public boolean deleteTransaction(String, Object[])
public boolean deleteTransaction(String, Object[], boolean)
public void executeSQL(String)
public void formatFieldValueByType(String, String, String)
public String getActiveConnection()
public Connection getCurrentConnection()
public int getKeyFieldMax(String, String)
public String insertAllTransaction(Erecord, String, String)
public String insertDefaultTransaction(Erecord, String, String)
public String insertTransaction(Erecord, String, String)
public RecordSet queryDataBase(String sql)
public ErecordSet selectTransaction(Object[], SQLField[], Object[], boolean)
public ErecordSet selectTransaction(Object[], SQLField[], Object[], boolean, SQLField)
public RecordSet selectTransactionRecordSet(Object[], SQLField[], Object[], boolean)
public RecordSet selectTransactionRecordSet(Object[], SQLField[], Object[], boolean, SQLField)
public void setActiveConnection(String)
public void setExclusiveOn()
public void setExclusiveOff()
public boolean updateBatchTransaction(ErecordSet, String, String[])
public boolean updateTransaction(Erecord, String tableName, Object[])
private void configFoxProDBMS(boolean, PrintWriter, Object[])
private ResultSet convertResultSetToERecordSet(ResultSet)
private ResultSet convertResultSetToERecordSet(ResultSet, SQLField[])
private Object[] extractTableNames(SQLField[])
private void loadTypeHash(String)
private void loadJdbcOdbcBridgeDriver()
private String removeDotNotation(String)
private ResultSet selectTransactionResultSet(Object[], SQLField[], Object[], boolean, SQLField)
```
3. UsrDefnRepConfig

```java
UsrDefnRepServerConfig extends PropertyResourceBundle

UsrDefnRepServerConfig()
```
Appendix E: **Serialized classes that are shared between JCR client applet and JCR server servlet.**

1. **ClientServerMsg**

   ```java
   ClientServerMsg implements java.io.Serializable
   
   private String methodName
   private Class[] parameterTypes
   private Object[] arguments
   private Object returnValue
   
   ClientServerMsg()
   ClientServerMsg(Object)
   ClientServerMsg(String,Object[])
   public String getMethodName()
   public Class[] getParameterTypes()
   public Object[] getArguments()
   public Object getReturnValue()
   public void setMethodName(String)
   public void setArguments(Object[])
   public void setReturnValue(Object)
   private void validateArguments(Object[])
   ```

2. **ERecord**

   ```java
   ERecord implements java.io.Serializable
   
   protected Hashtable record
   protected String keyField
   protected Vector orderedFieldSet
   
   public ERecord()
   public ERecord(Object[])
   public ERecord(Object[],String)
   public voidaddField(String)
   public voidaddField(String,String)
   public void removeField(String)
   public Object[] getFieldSet()
   public Object[] getOrderedFieldSet()
   public Boolean containsField(String)
   public String getKeyField()
   public void setKeyField(String)
   public String getFieldValue(String)
   public String getFieldValue(int)
   public Object[] getFieldValueArray()
   public void setFieldValue(String,String)
   public void printKeyValuePairs()
   ```
3. **SQLFilter**

```java
SQLFilter implements java.io.Serializable

private SQLField field
private String join_condition
private String comparison
private String value
private boolean filterValueType

public boolean FILTER_VALUE IS SUBQUERY
public boolean FILTER_VALUE IS FIELD
public boolean FILTER_VALUE IS NULL
public boolean FILTER_VALUE IS VALUE

public SQLFilter(String, String, String, String)
public String getJoinCondition()
public String getField()
public String getFieldTableName()
public String getFieldName()
public String getComparison()
public String getValue()
public boolean isFilterValueNonLiteral()

public void setJoinCondition(String)
public void setField(String)
public void setField(SQLField)
public void setComparison(String)
public void setValue(String)
public void setFilterValueType(boolean)
public String toString()
```

4. **SQLField**

```java
interface SQLField extends java.io.Serializable

public String getTableName()
public String getFieldName()
public String toString()
```
5. QueryStruct

QueryStruct implements java.io.Serializable

```
private int m_nQueryId;
private String m_strQueryDescription
private String m_strSQLCommand
private Vector m_vectorSavedFieldList
private Vector m_vectorReportFieldList
private Vector m_vectorFilterList
private String m_strSortOrder = "ASC"
private String m_strDELIMITER = ",,"
```

```
public QueryStruct(int, String, String, String)
public QueryStruct(int, String)
public QueryStruct(int, String, QueryStruct)
public int getQueryID()
public String getQueryDescription()
public String getQueryCommand()
public void setQueryID(int)
public void setQueryDescription(String)
public void setQueryCommand(String)
public Vector cloneFields()
public Vector cloneFilters()
public voidaddField(String)
public void addFieldAt(String, int)
public boolean removeField(String)
public void setFieldList(Object[])
public Object[] getReportFields()
public void setReportFields(Object[])
public void addFilter(String, String, String, String)
public SQLFilter getFilter(int)
public int getNumberOfFilters()
public void modifyFilter(int, String, String, String, String)
public String getFieldSortOrder()
public void removeFilter(int)
public Object[] getFieldList()
public Object[] getFilterListAsListItems()
public String getFieldListAsDelimitedString()
public String getFilterListAsDelimitedString()
private void parseFilterListStr(String strFilterList)
```
6. ERecordSet

ERecordSet implements java.io.Serializable

private Vector recordSet
private final int SORT_THRESHOLD = 7

ERecordSet()
ERecordSet(Object[])
ERecordSet(java.sql.ResultSet)
public void addRecord(ERecord)
public void insertRecordAt(ERecord, int)
public void removeRecord(ERecord)
public void removeAllRecords()
public ERecord returnRecordAt(int)
public void importRecords(ERecord[])
public ERecord[] exportRecords()
public Object clone()
public int recordCount()
public Object[][] getTabularData(Object[])
public void quickSortRecords(String, boolean, int, int)
public void insertionSortRecords(String, boolean, int, int)
private int compareTo(int, int)
private int compareTo(Date, Date)
private void swap(int, int)
private void printRecords()
Appendix F: Client class diagram.
null
Note:

- Uses
- Extends / Implements
- Contains
Appendix H: Software document.

A

B

C

ClientServerMsg

public String getMethodName(): this method returns the “service method name” that is requested to invoke by the server in a ClientServerMsg object.

public Class[] getParameterTypes(): this method returns the types of parameters in a ClientServerMsg object.

public Object[] getArguments(): this method returns the parameters in a ClientServerMsg object.

public Object getReturnValue(): this method returns the returned object from the server.

public setMethodName(String): this method sets the “service method name” in a ClientServerMsg object with a defined value.

public void setArguments(Object[]): this method sets the parameters in a ClientServerMsg object with defined values.

public void setReturnValue(Object): this method sets the returned object with a defined value.

private void validateArguments(Object[]): this method validates the parameters in a ClientServerMsg to see if their types are supported by the JCR.

abstract CommManager

public abstract Object sendServerRequest(String, Object[]): this method makes a server request to invoke a service call with parameter passing defined in an array Object. The server returned response is in the form of an Object.

public abstract Boolean testCommLink(): this method implements to verify if the client’s request to the server is successful.

CommManagerGroup

public void addCommManager(CommManager, String): this method implements to manage different CommManager instances.
public CommManager getCommManager(String): this method returns a CommManager instance.

public static HTTPCommManager createHTTPCommManager(URL): this method creates a HTTPCommManager instance for a server URL.

public static HTTPCommManager createHTTPCommManager(URL, AppletContext): this method creates a HTTPCommManager instance for a server URL and a client applet context.

public static RMICCommManager createRMICCommManager(String, String): this method creates a RMICCommManager instance for a host and a daemon.

public static RMICCommManager createRMICCommManager(String, String, String): this method creates a RMICCommManager instance for a host, a port number and a daemon.

D
DisplayCardListener

public void actionPerformed(ActionEvent): this method implements to display appropriate panel in card layout container based on user’s events.

E
ERecord

public void addField(String): this method adds a new field with empty string value to an ERecord object.

public void addField(String, String): this method adds a new field with a defined value to an ERecord object.

public void removeField(String): this method removes a field and its value in an ERecord object.

public Object[] getFieldSet(): this method returns the set of fields in an ERecord object.

public getOrderedFieldSet(): this method returns the set of fields that are in order in an ERecord object.

public int getNumberOfFields(): this method returns the number of fields in an ERecord object.

public int getKeyField(): this method returns the key field in an ERecord object.

public void setKeyField(): this method sets the key field in an ERecord object with a defined value.
public String getFieldValue(String): this method returns the field value of a field.

gpublic String getFieldValue(int): this method returns the field value of a field at a specific
index.

gpublic Object[] getFieldValueArray(): this method returns the values of all fields in a
form of an array Object.

gpublic setFieldValue(String, String): this method sets the field value for a field.

gpublic void printKeyValuePairs(): this method displays all the fields and their values in
the default output (it often is the terminal screen).

ERecordSet

gpublic void addRecord(ERecord): this method adds a new record to an ERecordSet
object.

gpublic void insertRecordAt(int): this method inserts a new record to an ERecordSet
object at a specific index.

gpublic void removeRecord(int): this method removes a record from an ERecordSet object
at a specific index.

gpublic void removeAllRecords(): this method removes all the records in an ERecordSet
object.

gpublic ERecord returnRecordAt(int): this method returns a record object at a specific
index.

gpublic void importRecords(ERecord[]): this method imports an array of records into an
ERecordSet object.

gpublic ERecord[] exportRecords(): this method exports all records in an ERecordSet
object.

gpublic Object clone(): this method clones the ERecordSet object.

gpublic int recordCounts(): this method returns the number of records in an ERecordSet
object.

gpublic Object[][] getTabularData(Object[]): this method returns data of an ERecordSet
object in a form of Object[][] so that the data can be conveniently used to display in a
java table object.
public Object[][] getColumnarData(Object[]): this method is deprecated and replaced by getTabularData(Object[]).

class public void QuickSortRecords(String, boolean, int, int): this method sorts the records from a specific index to a specific index in an ERecordSet object ascending or descending using “Quick Sort” algorithm.

class public void InsertionSortRecords(String, boolean, int, int): this method sorts the records from a specific index to a specific index in an ERecordSet object ascending or descending using “Insertion Sort” algorithm.

private int compareTo(String, String): this method compares two values (the values could be string, int or date object).

private int compareTo(int, int): this method compares two integers.

private int compareTo(Date, Date): this method compares two dates.

private void printRecords(String): this method displays all the values of a defined field in an ERecordSet object in the default output for debugging purpose.

**F**

**FoxproDBMS**

public void connectToDataSource(String) throws SQLException: this method implements to connect to an user-configured database source name (DSN).

public String getActiveConnection(): this method returns the active database source name.

public String setActiveConnection(String): this method sets the active database source name to a defined database source name.

public void closeConnection(String) throws SQLException: this method releases the database connection.

public void closeAllConnection() throws SQLException: this method releases all database connections.

public Connection getCurrentConnection(): this method returns the current database connection.

public int getKeyFieldMax(String, String): this method returns the maximum value in a field of a database table.
public String formatFieldValueByType(String, String, String) throws SQLException: this method formats field value according to its database field type.

public int setExclusiveOn(): this method sets “exclusive” option for database transactions that require to be in “exclusive” mode (for example, insert transaction).

public int setExclusiveOff(): this method turns off “exclusive” mode.

public ERecordSet queryDatabase(String): this method performs a database select transaction with a defined SQL statement.

public int executeSQL(String): this method executes a defined SQL statement.

public ERecordSet selectTransaction(Object[], SQLField, Object, Boolean): this method performs a database select transaction with defined database tables, fields, filters and distinct/all option.

public RecordSet selectTransaction(Object[], SQLField, Object, Boolean): this method functions as the same as above but returns a RecordSet instead of a ERecordSet.

public ERecordSet selectTransaction(Object[], SQLField, Object, Boolean, SQLField): this method performs a database select transaction with defined database tables, fields, filters, distinct/all option and sort order.

public RecordSet selectTransaction(Object[], SQLField, Object, Boolean, SQLField): this method performs as the same as above but returns a RecordSet instead of a ERecordSet.

public String insertDefaultTransaction(ERecord, String, String): this method performs a database insert transaction with a defined record, database table and database table key field.

public String insertAllTransaction(ERecord, String, String): this method is deprecated and replaced by insertDefaultTransaction(ERecord, String, String).

public String insertTransaction(ERecord, String, String): this method is deprecated and replaced by insertDefaultTransaction(ERecord, String, String).

public boolean updateTransaction(ERecord, String, Object[]): this method performs a database update transaction with a defined record, database table and filters.

public boolean updateBatchTransaction(ERecordSet, String, String[]): this method performs a database update transaction with a defined record set, database table and key fields.

public boolean deleteTransaction(String, Object[]): this method performs a database delete transaction with a defined database table and filters.
public boolean deleteTransaction(String, Object[], boolean): this method performs a
database delete transaction with a defined database table and filters. After deletion, all the
marked-delete records will also be permanently removed out of the database to save
space. This method requires being in exclusive mode to perform.

private void loadJdbcOdbcBridgeDriver(): this method loads database driver dynamic
link library(dll) into operating memory.

private void configureFoxProDBMS(boolean, PrintWriter, Object[]): this method
configures the FoxproDBMS instance to connect to a defined DSN and whether to log
database transactions.

private void loadFieldTypHash(String): this method finds out what database table field
types for a specific database table.

private Object[] extractTableNames(SQLField[]): this method extracts database table
names from database table fields in the form of SQLField.

private String rmDotNotation(String): this method returns the field name from field name
that is in the form of “tablename.fieldname”.

private ResultSet
selectTransactionResultSet(Object[], SQLField[], Object[], boolean, SQLField): this
method performs a database select transaction with defined tables, fields, filters,
distinct/all option and sort order field.

private ERecordSet convertResultSetToERecordSet(ResultSet): this method converts a
ResultSet instance into a ERecordSet instance.

private ERecordSet convertResultSetToERecordSet(ResultSet, SQLField[]): this method
converts a ResultSet instance into a ERecordSet instance with defined fields.

G
interface GUIInterface

H
HTTPCommManager

public Object sendServerRequest(String, Object[]): this method makes a server request to
invoke a service call with parameters defined in an array Object. The server returned
response is in the form of an Object.

public Object sendServerPOSTRequest(String, Object[]): this method makes a server
POST request to invoke a service call with parameters defined in an array of Object.
public InputStream sendSeverGETRequest_APPLET(Properties): this method makes a server GET request with defined arguments in the form of a Properties instance.

public InputStream sendSeverGETRequest_APPLET(String): this method makes a server GET request with defined arguments in the form of a String instance.

public InputStream sendSeverGETRequest_BROWSER(Properties): this method makes a server GET request with defined arguments in the form of a Properties instance. The returned result is displayed in a new browser window.

public InputStream sendSeverGETRequest_BROWSER(Properties, String): this method makes a server GET request with defined arguments in the form of a Properties instance. The returned result is displayed in a new browser window with a defined target.

public InputStream sendSeverGETRequest_BROWSER(String): this method makes a server GET request with defined arguments in the form of a String instance. The returned result is displayed in a new browser window.

public InputStream sendSeverGETRequest_BROWSER(String, String): this method makes a server GET request with defined arguments in the form of a String instance. The returned result is displayed in a new browser window with a defined target.

public Boolean testCommLink(): this method implements to verify if the client’s request to server is successful.

**HTTPMessage**

public InputStream sendGetMessage(Properties) throws IOException: this method sends a server GET request with defined arguments in the form of a Properties instance.

public InputStream sendPostMessage(Serializable) throws IOException: this method sends a server POST request with defined arguments in the form of a Serializable object.

public String toEncodedString(Properties): this method converts arguments in the form of Properties object into URL-encoded string.

**ImageManager**

public void add(String, URL): this method implements to manage images that are located at a specific URL.

public Image get(String): this method returns requested image.

**IniBundle**
public Enumeration getKeys(String): this method returns all the keys of a property group.

public String get.osgi(String): this method returns the value for a key.

public String get.osgi(String, String): this method returns the value for a key in a property group.

**IntroPanel**

public void run(): this method displays the JCR Html introduction page in the introduction panel.

private void buildIntroPanelGUI(): this method constructs the introduction panel GUI.

**OutputSortOrderPanel**

public static OutputSortOrderPanel getInstance(): this method returns the instance of OutputSortOrderPanel class.

public Object[] get.osgiFieldsNSSortOrder(): this method returns query output fields and sort orders.

public Object[] get.osgiFieldsNSSortOrderDESC(): this method returns the English descriptions of query output fields and sort orders.

public void reviewQueryInfo(int): this method detects changes in queries to perform appropriate actions.

**QueryManager**

public void run(): this method sends an appropriate database insert, delete, update or select transaction to the server.
public void loadQuerySet() throws Exception: this method loads query field ids and query field English descriptions from the database.

public void printFieldDescriptionNameHashtables(): this method prints out query field ids and query field descriptions for debugging.

public Object[] getFieldDescriptionSet(): this method returns descriptions of all query fields from the database.

public String getFieldDescriptionFor(String): this method returns a field description for a field.

public Object[] getFieldNameSet(): this method returns all query field ids.

public String getFieldNameFor(String): this method returns field id for a field description.

public int getNumberOfQueries(): this method returns the number of queries in the database.

public Object[] getQueryDescriptions(): this method return descriptions of all queries in the database.

public String copyQuery(int): this method copies a query at a specific index into a new query.

public String copyQuery(int,String): this method copies query at a specific index into a new query with a new description.

public void createQuery(QueryStruct): this method creates a new query from a QueryStruct object.

public void deleteQuery(int): this method deletes a query at a specific index.

public void renameQuery(int,String): this method renames a query at a specific index.

public void selectQuery(int): this methods marks the query at a specific index as selected.

public void selectQueries(): this method loads all the queries from the database.

public String printFields(): this method returns all the fields of a query.

public Object[] restoreFieldList(): this method restores the previous fields of a query.

public void saveFieldList(Object[]): this method saves current fields of a query.
public void addFilter(String, String, String, String): this method adds a filter defined by a
field condition, a field, a field comparison and a field value to a query.

public void deleteFilter(int): this method deletes a filter of query at a specific index.

public Object[] getFilterList(): this method returns the list of filters of the currently
selected query.

public int getNumberOfFilters(): this method returns the number of filters of the currently
selected query.

public String getFilterCondition(int): this method returns the filter condition of a query at
a specific index.

public String getFilterField(int): this method returns filter fields of a query at a specific
index.

public String getFilterComparison(int): this method returns filter comparisons of a query
at a specific index.

public String getFilterValue(int): this method returns filter values of a query at a specific
index.

public void modifyFilter(int, String, String, String, String): this method modifies a filter of
a query at specific index with a defined field condition, a defined field, a defined field
comparison and a defined field value.

public String getFieldSortOrder(): this method returns the current sort order field of the
currently selected query.

public void setFieldOrder(String): this method sets sort order of the currently selected
query to a defined value.

public String printFilters(): this method returns the filter list of the currently selected
query.

public QueryManagerListener addListener(QueryManagerListener): this method
manages QueryManagerListener listeners.

public void removeListener(QueryManagerListener): this method removes a
QueryManagerListener listener.

public void notifyListeners(): this method notifies all the currently registered
QueryManagerListener listeners.
public void generateReport(): this method runs the selected query and displays result to user.

public void showStudentReport(): this method displays student detail report in a new browser window.

public void showQueryReport(): this method displays query result in a new browser window.

private void loadFieldSet() throws Exception: this method loads the English descriptions of all queries from the database.

interface QueryManagerListener

public void reviewQueryInfo(int): this method detects changes in queries to perform appropriate actions.

QueryPanel

public static QueryPanel getInstance(): this method returns the instance of QueryPanel class.

public reviewQueryInfo(int): this method detects changes in queries and performs appropriate actions.

private void buildQueryPanelGUI(): this method calls appropriate methods to construct the “Query Console” panel GUI.

private void buildQueryPanelTitle(): this method builds the title label of the “Query Console” panel.

private JPanel buildQueryListPanel(): this method builds the top part of the “Query Console” panel.

private JPanel buildCreateQueryPanel(): this method builds the bottom part of the “Query Console” panel.

QueryStruct

public int getQueryID(): this method returns the id of a query.

public String getQueryDescription(): this method returns the name of a query.

public String getQueryCommand(): this method returns the query “Insert”, “Update”, “Select” or “Delete” command.
public void setQueryID(int): this method sets the query id of a query with a defined value.

public void setQueryDescription(String): this method sets the query name of a query with a defined value.

public void setQueryCommand(String): this method sets the query command of a query with a defined value.

public Vector cloneFields(): this method clones all the query fields.

public Vector cloneFilters(): this method clones all the query filters.

public void addField(String): this method adds a new field with a defined value to a query.

public void addFieldAt(String,int): this method inserts a new field with a defined value at specific index.

public boolean removeField(String): this method removes a field out of a query.

public void setFieldList(Object[]): this method sets the field list with defined values.

public Object[] getReportFields(): this method returns the list of fields that will be included in the report.

public void setReportFields(Object[]): this method sets the list of fields that will be included in the report with defined values.

public void addFilter(String,String,String): this method adds new filter with a defined field condition, a defined field comparison and a defined field value to a query.

public SQLFilter getFilter(int): this method returns the query filter at a specific index.

public int getNumberOfFilters(): this method returns the number of filters in a query.

public void modifyFilter(int,String,String,String,String): this method modifies a query filter at a specific index.

public void removeFilter(int): this method removes a query filter at a specific index.

public String getFieldSortOrder(): this method returns the sort order field of a query.

public void setFieldSortOrder(String): this method sets the sort order field of a query with a defined value.
public Object[] getFieldList(): this method returns the field list of a query.

public Object[] getFilterListAsListItems(): this method returns an array of individual filters in a query.

public String getFieldListAsDelimitedString(): this method returns the list of fields in a query in a form that every field is separated by a defined delimiter.

public String getFilterListAsDelimitedString(): this method returns the list of filters in a query in a form that every filter is separated by a defined delimiter.

class RecordFilterPanel

public RecordFilterPanel getInstance(): this method returns the instance of the RecordFilterPanel class.

class RecordFilterPanel

public void reviewQueryInfo(int): this method detects changes in queries to perform appropriate actions.

private void buildRecordFilterPanelGUI(): this method calls appropriate methods to construct the “Enter Record Filters” panel GUI.

private JLabel buildRecordFilterPanelTitle(): this method builds the title label of the “Enter Record Filters” panel.

private JPanel buildFilterFieldPanel(): this method builds the top part of the “Enter Record Filters” panel.

private JPanel buildFilterListPanel(): this method builds the bottom part of the “Enter Record Filters” panel.

private DefaultListModel buildListModel(Object[]): this method returns a DefaultListModel from a defined array of items.

private AbstractButton buttonProps(AbstractButton,Color): this method customizes a button with a defined color.

private AbstractButton buttonProps(AbstractButton,Font,Color,int,int): this method customizes a button with a defined font, a color and a size.
private String removeCommas(String): this method removes commas in a string.

**ReportPanel**

public static ReportPanel getInstance(): this method returns the instance of the ReportPanel class.

public void buildReportPanelGUI(): this method calls appropriate methods to construct the “Custom Report” panel GUI.

public void sortByColumn(int): this method sorts data in a table column ascending or descending for a specific column.

public void viewReport(ERecordSet): this method displays query result data in the form of a RecordSet object in the “Custom Report” panel table.

public void viewReport(String[][]): this method displays query result data in the form of String[][] in the “Custom Report” panel table.

public void mouseClicked(MouseEvent): this method handles user event “mouse click” on the “Custom Report” panel table.

public void mouseEntered(MouseEvent): this method handles user event “mouse enter” on the “Custom Report” panel table.

public void mouseExited(MouseEvent): this method handles user event “mouse exit” on the “Custom Report” panel table.

public void mousePressed(MouseEvent): this method handles user event “mouse press” on the “Custom Report” panel table.

public void mouseReleased(MouseEvent): this method handles user event “mouse release” on the “Custom Report” panel table.

private void buildReportPanelTitle(): this method builds the title label of the “Custom Report” panel.

private JPanel buildReportPanelGUI(): this method builds the main panel of the “Custom Report” panel.

private displayReport(Object[], Object[][]): this method displays query result data in the form of Object[][] in the “Custom Report” panel table that has its headers defined by an array Object.
private String removeDotNotation(String): this returns field name from the field name
that has the form “tablename.fieldname”.

interface SQLField

public String getTableName(): this method returns the database table name.

public String getFieldName(): this method returns the database table field name.

public String toString(): this method returns the full name of a SQLField instance.

SQLFilter

public String getJoinCondition(): this method returns the join condition (AND, OR or
empty value) of a filter.

public String getField(): this method returns the full name of a filter.

public String getTableName(): this method returns the database table name.

public String getFieldName(): this method returns the database table field name.

public String getComparison(): this method returns the comparison value of a filter.

public String getValue(): this method returns the value of the filter value field in a filter.

public boolean isFilterValueNonLiteral(): this method determines whether the value of
the filter value field in a filter is literal or not.

public void setJoinCondition(String): this method sets the join condition of a filter with a
defined value.

public void setField(String): this method sets the field of a filter with a defined value.

public void setField(SQLField): this method sets the field of a filter with a SQLField
object.

public void setComparison(): this method sets the comparison value of a filter with a
defined value.

public void setValue(String): this method sets the value of the filter value field in a filter
with a defined value.

public void setFilterValueType(boolean): this method sets the filter type to be literal or
non-literal.
public void toString(): this method returns the full string value of a filter.

public void getFilterAsDelimitedString(): this method returns the string value of a filter in a form that its filter field, its filter join condition, its filter comparison, its filter field value are separated by a defined delimiter.

public String getFilterAsListItem(): this method returns the string value of a filter in a form that blank space is padded into the filter to separate where it is suitable.

private void parsefilterStr(String): this method parses a string into a filter.

T

U

UDRComboBox

public void setEnabled(boolean): this method sets UDRComboBox instance enabled (UDRComboBox is able to be selected by user or to receive user event).

public String getSelectedItem(): this method returns the value of selected item in the UDRComboBox instance.

public String setSelectedItem(String): this method sets which item in the UDRComboBox instance selected.

public void Opaque(boolean): this method sets UDRComboBox instance to whether the Java Graphic object will paint UDRComboBox at every single pixel or not.

UDRTextField

public String getText(): this method returns text value of the UDRTextField instance.

public void setText(String): this method sets text value of the UDRTextField with a defined value.

public void setEnabled(boolean): this method sets a UDRTextField instance enabled.

public void setOpaque(boolean): this method sets UDRTextField instance to whether the Java Graphic object will paint UDRTextField at every single pixel or not.

UsrDefnRepApplet

public void init(): this method initializes the applet when it is first loaded.
public static UsrDefnRepApplet getApplet(): this method returns the instance of the UsrDefnRepApplet class.

public static ImageManager getImageManager(): this method returns the ImageManager instance.

public static IniBundle getIniBundle(): this method returns the IniBundle instance.

public static CommManagerGroup getCommManagerGroup(): this method returns the CommManagerGroup instance.

public static AppletContext getEnvironment(): this method returns the AppletContext instance.

public static String getVersion(): this method returns the version of the JCR.

UsrDefnRepGUIManager

public static JPanel getPanelDesktop(): this method returns the main panel of the JCR.

public static JToggleButton getReportButton(): this method returns the instance of the “Custom Report” button.

private void buildUsrDefnRepGUI(URL,URL): this method calls appropriate methods to construct the GUI for the JCR.

private JToolBar buildButtonBar(URL,URL): this method builds the tool bar of the JCR.

private JPanel buildDesktopPanel(): this method builds the main panel of the JCR.

private AbstractButton buttonProps(AbstractButton,Color): this method customizes a button with a color.

private AbstractButton buttonProps(AbstractButton,Font,Color,int,int): this method customizes a button with a font, a color and a size.

interface UsrDefnRepInterface

UserDefnRepServer

public void init(ServletConfig) throws ServletException: this method initializes the JCR server servlet.

public void doGet(HttpServletRequest,HttpServletResponse) throws ServletException, IOException: this method processes Http GET request.
public void doPost(HttpServletRequest, HttpServletResponse) throws ServletException, IOException: this method processes Http POST request.

public void destroy(): this method releases all the used resources when the servlet web server stops.

public Vector queryStudentReportFieldDesc() throws SQLException: this method returns all the query fields from the database.

public Vector select_UDR_Queries() throws SQLException: this method returns all the queries from the database.

public String insert_UDR_Query(String, String, String): this method inserts a new query with a defined query name, query fields and query filters into the database.

public Boolean delete_UDR_Query(String): this method deletes a query with specific query id from the database.

public Boolean update_UDR_Query(String, String, String, String): this method updates a query of a specific query id in the database with a defined query name, query fields and query filters.

public Hashtable run_UDR_Query(QueryStruct) throws SQLException: this method runs a query and returns the result in a Hashtable instance.

private int getNumberOfRecordsQuery(QueryStruct) throws SQLException: this method returns the number of records from the result of running a query.

private String buildRunReportSQL(QueryStruct) throws SQLException: this method builds the SQL statement from a QueryStruct instance.

private ERecord getStudentInfo(String) throws SQLException: this method returns a student detail record from a defined student id.

private void printStudentReport(PrintWriter, String) throws SQLException: this method displays a student detail record of a specific student id in a new browser window.

private void printError(PrintWriter): this method displays error messages to user if anything goes wrong when server servlet processes client applet’s requests.

private void printQueryReport(Integer, String, PrintWriter): this method displays query result in a new browser window.

private void handleUpdate(String, String, String): this method performs a database update transaction for a SQL statement case that the statement’s length is smaller than 250.
private void handle250Update(String, String, String): this method performs a database update transaction for a SQL statement case that the statement’s length is greater than or equal to 250. This method is implemented due to the reason that the Open Database Connectivity (ODBC) Foxpro driver cuts off at 255 characters in length for any SQL statement. ‘250’ is selected rather than ‘255’ since ‘250’ is nicely round and safer.
Appendix I: **Edulog Coding Standards in the Java Context.**

**Naming conventions**
Classes should be mixed case, no underscores.

```java
public class ColorGenerator
...
```

Class names can be prefaced with packaging info (e.g. "JMC" below)

```java
public class JMCGeoObject
...
```

Variables should be mixed case and prefaced with type information (that is, use Hungarian notation). Exceptions to this are constants (see the 'constants' section below) and trivial loop counters (where i/j/k are typically used).

```java
boolean bIsShown, bWasModified;
short i, nNumColors;
int nNumStudents;
long lNumSiteHits;
char chKeyHit;
float fRatio;
double dDistance;
String strName;
Point pointCenter;
Rectangle rectZoomBox;
Color colorChosen;
Hashtable hashStudents;
Stack stackPrevViews;
Color [] arrayColors;
Vector vectorPoints;
```

Single-use local variables, or arguments, can be trivially named (of type information only).

```java
Color color;
public setCenterPoint(Point point)
...
public setAttribute(JMCAtribute attribute)
...
```

Member variables (excluding constants) should be prefaced with an "m_". This is in addition to the Hungarian notation.
private HashSet m_hashListeners;
private JMCCoordMap m_coordmap;
private int m_nNumPoints;
private Color m_color;

Methods should be named for the actions they perform, with the verb beginning the method name in lower case, followed by the object of the action in mixed case. Accessor methods should use "get" and "set".

public int getSize()
...
public void setSelectionAttribute(JMCAtribute attribute)
...
private void fireChangedEvent()
...
public void showItem(int nIndex)
...
public Image composeImage(Rectangle rect ...)
...

Constants
Constants should be static final member variables, specified in mixed case only (non-Hungarian). Their scope (public, protected, private) should reflect who uses them. Their usage always should be with the class name specified for clarity.

public class JMCUnits
{
    static public final short Inches = 0;
    static public final short Feet  = 1;
    static public final short Yards = 2;
    ...

    // sample wrong usage
    if (nUnits == Inches)
    ...

    // sample correct usage
    if (nUnits == JMCUnits.Inches)
    ...

Comments
Comments should be treated just like code, as they should be maintained just like code. Isolate comments just as you would isolate code. Avoid cut-and-paste of comments (and code, for that matter).
Don't comment the obvious. Use comments sparingly and judiciously so that when a comment is seen, it's noticed and read. Comment the exception, not the general case, and avoid exceptions when you can. Write your code so it documents itself.

```
// set the cursor <- bad comment
m_list.setCursor( ... );
```

Don't comment the language. If you feel the need to do so, your construct is probably too long and needs to be refined.

```
// four bad comments
for (...) // loop through items
{
  if (i < nMaxItems) // if i is in acceptable range
  {
    ...
  } // end of if
} // end of for loop
```

**Declarations**

Variable declarations should be at the beginning of the code block (i.e. right after the opening brace). In-code declarations should be restricted to constructors that depend on the execution of some preceding code.

Declaration of a loop counter in the initialization section of a for statement (i.e. `for (int i = 0; i < nNumItems; i++)`) should be restricted to cases when the variable is used only inside the for loop and in no other for loops in the code block.

Grouping of declarations should be most importantly to signify related usage, then by type.

Initialization in the declaration should be restricted to primitive types and should not call a function.

**Conditions**

Use non-operated testing on boolean's (i.e. don't test `== true`).

```
// wrong
if (blsEnabled == true)
...

// right
if (blsEnabled)
...
```
Control flow
Avoid using labels (essentially goto's), period. Java allows the "break" and "continue" statements to take a single argument -- a source code label. Don't use them ... structure your control flow properly.

Avoid returns from anywhere but at the end of a method.

Spacing & style
Indentation should be two spaces. Avoid using tabs unless your editor can convert tabs to spaces correctly when saving.

Braces should be on a line by themselves, directly under the preceding clause (i.e. not indented).

Braces should be used when the condition clause spans multiple lines.

Avoid putting multiple statements on the same line, except in trivial parallel cases to emphasize the parallel nature. And in those cases, line up the parallel statements vertically.

```java
switch (nSize)
{
    case Shirt.Small:  m_nNumInBox = 100; break;
    case Shirt.Medium: m_nNumInBox = 70; break;
    case Shirt_LARGE:  m_nNumInBox = 40; break;
}
```

One space after semicolons in for statements.

```
// wrong
for (i = 0; i < nNumItems; i++)
    ...

// right
for (i = 0; i < nNumItems; i++)
    ...
```

One space around all binary and assignment operators. More than one space can be used to add readability when appropriate.

```
// wrong
if ((item!=null)&&item.isEnabled())
    nTotal+=item.getNumAttached();
```
// right
if ((item != null) && item.isEnabled())
    nTotal += item.getNumAttached();

// more readable
rect.width = Math.max(rectExtents.width, rect.width);
rect.height = Math.max(rectExtents.height, rect.height);

No spaces between unary operators and their operands.

// wrong
nNumStops ++;

// right
nNumStops++;

Blank lines:
-- always exactly one between declarations and code
-- always at least one between functions
-- exactly one to emphasize naturally grouped statements as appropriate
-- never immediately following an opening brace or left paren
-- never immediately preceding a closing brace or right paren

Imbedded assignments should be used carefully, and always enclosed in parens.

// wrong
if (nNumItems = item.getNumItems() > 0)
    ...

// right
if ((nNumItems = item.getNumItems()) > 0)
    ...

Type casts should use prefix notation. There should not be a space between the closing
paren and the object of the cast (e.g. '(int)nValue', '(int) Value').

Argument lists that span multiple lines, should line up vertically under the first
argument. Conditions that span multiple lines should line up vertically as well.

bOkay = geoobject.draw((Graphics2D)g, coordmap,
    item.getAttribute());

if ((item != null) &&
    (item.getNumAttached() > 0) &&
    buttonRemove.isEnabled())
\begin{verbatim}
{
...
}
for (point = geoobject.getFirstPoint();
    point != null;
    point = geoobject.getNextPoint())
{
    ...
}
\end{verbatim}

**Statement spacing & style**

The 'for' statement

\begin{verbatim}
for (i = 0; i < nNumItems; i++)
{
}
\end{verbatim}

The 'if-else if-else' statement

\begin{verbatim}
if (i < nNumItems)
{
}
else if (i == nNumItems)
{
}
else
{
}
\end{verbatim}

The 'while' statement

\begin{verbatim}
while (i < nNumItems)
{
}
\end{verbatim}

The 'do-while' statement

\begin{verbatim}
do
{
}
while (i < nNumItems);
\end{verbatim}

The 'switch' statement

Use of the "drop-through" mechanism in switch statements should be used carefully, and not in lieu of extracting what should be a function. Should local scope be needed in a case, the braces should be in line with the case, the statements indented two spaces, and the break should be inside the braces (e.g. the default case below).
switch (nType)
{
    case SMALL:
        nSmallCount++;
        break;

    case MEDIUM:
        nMediumCount++;
        break;

    case LARGE:
        nLargeCount++;
        break;

    default:
    {
        nIllegalTypeCount++;
        System.out.println("another illegal one");
        break;
    }
}

The 'return' statement
The use of returns from places other than the end of a function should be restricted to "bailing out" due to an error condition. Avoid using a return in place of good logic.

Methods that are 'void' should not have a 'return;' before the closing brace.

return bOkay;
return (i < nNumItems);
return (object != null);
return item.getNumAttached();

The '?:' statement

Access
Avoid public member variables. Provide access methods to protect your members. Adding functional access to even the most trivial member variable leaves the option open for making it less-trivial in one place (instead of everywhere it's used) in the future.

// wrong
class Segment
... public float m_fX0,m_fY0,m_fX1,m_fY1;
...

// wrong's usage
segment.m_fX0 = 3.14159f;
if (segment.m_fY0 > segment.m_fY1)
...

// correct
class public Segment
{
...
public void setX0(float fx)
{
    m_fX0 = fx;
}
public float getX0()
{
    return m_fX0;
}
...
}

// correct's usage
segment.setX0(3.14159f);
if (segment.getY0() > segment.getY1())
...

Reference arguments
Be extremely careful passing object references around. Java has regressed to a Fortran-esque style of allowing only reference passes of objects, and this can bite you if you're not clear on it. When the caller's object needs to be protected, he should pass a copy. And when appropriate, the callee should make a copy of the passed in object. For example, JMCCoordmap makes a copy of the passed in Rectangle objects.