REVERSE ENGINEERING: JAVA CODE TO UML DIAGRAM SHOWING DEPENDENCIES

Enrol. No. - 101258

Name of Student - ANKIT RAJPUT

Name of supervisor(s) - MR. RAVINDARA BHATT

JAN 2014 – MAY 2014

Submitted in partial fulfilment of the Degree of

Bachelor of Technology

in

Computer Science Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & INFORMATION TECHNOLOGY

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY WAKNAGHAT

STUDENT DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institutes of higher learning, except where due acknowledgement has been made in the text.

Place: Signature:

Date: Name:

Enrollment No.:

CERTIFICATE

This is to certify that the project entitled **"Reverse Engineering : Java code to uml diagram showing dependencies "** done by **Ankit Rajput** (101258) is an authentic work carried out by him at JUIT, during his 8th semester period from Jan'15 2014 to May'9 2014, under my guidance. This project is submitted in partial fulfilment for the award of the degree of Bachelor of Technology (Computer Science & Engineering) from Jaypee University of Information Technology, Solan. The matter embodied in this project has not been submitted earlier for award of any degree or diploma to the best of my knowledge and belief.

Signature of the Supervisor:

Name of the Supervisor : Mr. Ravindara Bhatt

Place: Jaypee University of Information Technology

Solan, Himachal Pradesh

Date:

ACKNOWLEDGEMENT

This is a great opportunity to acknowledge and to thank all those persons without whom this project would have been impossible. I would like to add a few heartfelt words for the people who were a part of this project in numerous ways. I was extremely grateful to Mr. Ravindara Bhatt, for his indefatigable guidance, valuable suggestion, moral support, constant encouragement and contribution of time for the successful completion of project work. I am very thankful to him, for providing all the facilities needed during the project development.

I would also like to extend my sincere thanks to all the staff members of the JUIT for providing us with different facilities such as computer support and library infrastructure support etc. required for the project work during our stay at the institute.

Signature of Student

Name: Ankit Rajput

Date :

Signature of Supervisor

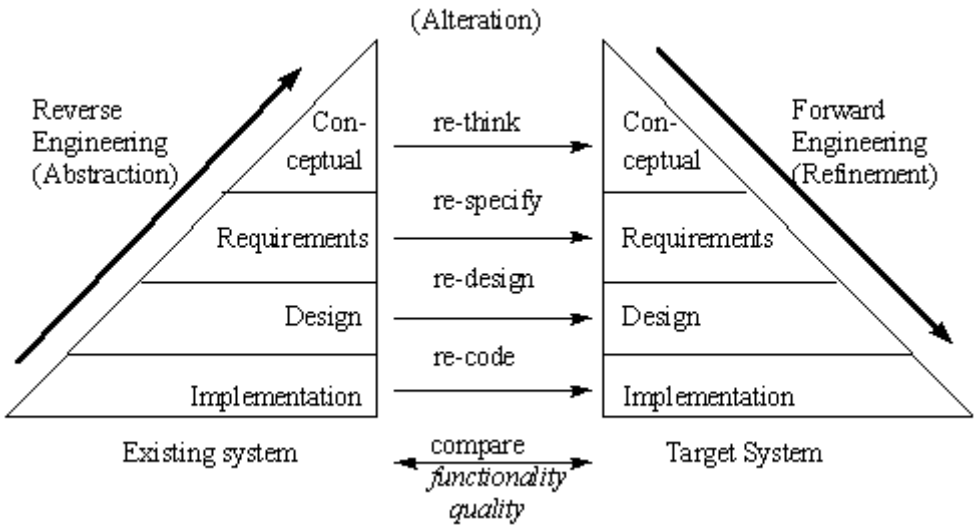
Name: Ravindara Bhatt

Date:

ABSTRACT:

Software Engineering research and industry recognize the need for practical tools to support reverse engineering activities. Most of the well-known CASE-tools nowadays support reverse engineering in some way or other. Reverse engineering is first step towards software Architecture recovery. The most commonly used standard today is Unified Modeling Language to depict the architecture and design of an application. An UML class diagram describes the architecture of object oriented programs. Class diagram captures the essence of its design. Most of the existing systems do not have reliable software architecture and some legacy systems are designed without software architecture design phase. By using reverse engineering tools we can generate class diagram as part of software architecture recovery.

This tool assess capabilities of software reverse engineering tools to generate class diagram from java source code.



LIST OF FIGURES :

TABLE OF CONTENTS :

CHAPTER 1: INTRODUCTION

PROBLEM STATEMENT

In the world of computing applications, approximately 30-35% of the overall total lifecycle costs are devoted to helping the programmer understand the functionality of existing code. This is a necessary task, in order to correctly make required changes in response to new requirements, to resolve errors, or perform other changes [Tomic94]. A thorough understanding of the logic, design, and structure of existing code will help developers, management, and analysts more accurately estimate the maintenance and enhancement costs, analyze code complexity, undertake thorough testing, and estimate software reliability more effectively and efficiently. However, with the "time is money" mentality that dominates in most workplaces, a professional is rarely given a sufficient amount of time to thoroughly and comprehensively complete a task in a manner that does not introduce additional problems in the software.
Reverse engineering, analyses system's code, documentation and behaviour to create system abstractions and design information . Reverse Engineering is , essentially, the practice of examining existing systems, at any stage, to identify elements and dependencies. This information is then used to gain more knowledge about the design, the structure, system code, and functionality.

There are many existing tools, such as Rational Rose®, jGRASP®, NetBeans®, and Eclipse® , that provide a degree of reverse engineering. Several tools and frameworks take Java code as input and generate the Unified Modeling Language (UML) class diagrams. These diagrams are helpful to the users by illustrating object dependencies; however, they tend to be high level and leave much to be desired about "lower level" (i.e., code level) application specifics. While object dependencies are indicated through UML associations, multiplicity, direction, and other real-world objects can be complex. General dependencies at this architectural level (class diagrams with dependencies) are helpful, but such renderings leave the professional in dire need of much more detailed analysis of object dependencies extending down to method-level dependencies, which is where actual code maintenance will occur.

PROJECT GOAL / MOTIVATION:

As a developer, it would be more beneficial to have a framework that drills down a level further than providing high-level class dependencies. A comprehensive reverse engineering framework that, when given an unknown Java program, will analyze the existing structural characteristics and generate detailed low-level dependencies and relationships among code segments would be helpful in a workplace environment.
The framework would, by class, show all methods declared in the class and what methods they invoke. It would also, by each method, show the class and methods it is referenced by. Equivalently, "who" invokes the services of this class and what services of other classes does "this" class invoke would be shown.
While this is clearly an arduous undertaking, a framework that provides this level of analysis up front to a software professional before starting a software maintenance task has multiple benefits. It should assist the user in both understanding of the design and complexity of an existing application as well as assuring the user a more reliable maintenance undertaking.
To set the stage for this undertaking, this documentation first presents a number of popular development frameworks containing reverse engineering tools, such as Rational Rose®, jGRASP®, NetBeans®, Eclipse® and others, in order to comprehensively identify both their strengths and shortcomings. The thesis will then present the details of the new framework that provides detailed method dependencies and associations.

ORGANIZATION OF THE REPORT :

It has been organized in the following manner : The present chapter gives an introduction and explains about the statement of the problem . Chapter 2 gives an overview of the literature survey. Chapter 3 gives the details of the implementation of the tool. Chapter 4 explains the results of the project . Appendix includes the project code and references.

CHAPTER 2: REVIEW OF THE LITERATURE

2.1 Reverse Engineering

2.1.1 Related Areas and Sub-Topics in Reverse Engineering

Reverse engineering is a broad subject area, which includes a variety of sub-topics and components. Many terms are used when discussing reverse engineering.

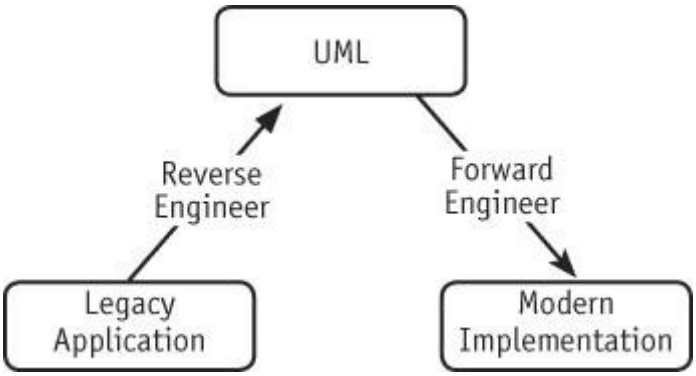
• Forward Engineering - the process of starting at the gathering of requirements and then following through to design and finally to the implementation of the application.

• Design Recovery - gathering additional information, like domain knowledge, outside information, and deductive information for inclusion with other observations, to assist the professional in better understanding of the system being studied.

• Restructuring - the movement from one form to another form at the same level of abstraction without changing the system's output. Essentially, it is changing code to put it in a more structured format.

• Reengineering - the investigation and modification of a system to rebuild it in a new form. It is usually accomplished by reverse engineering a systemand then forward engineering the system.

• Software Maintenance - includes changing source code to correct errors, improve performance, fix problems, etc.

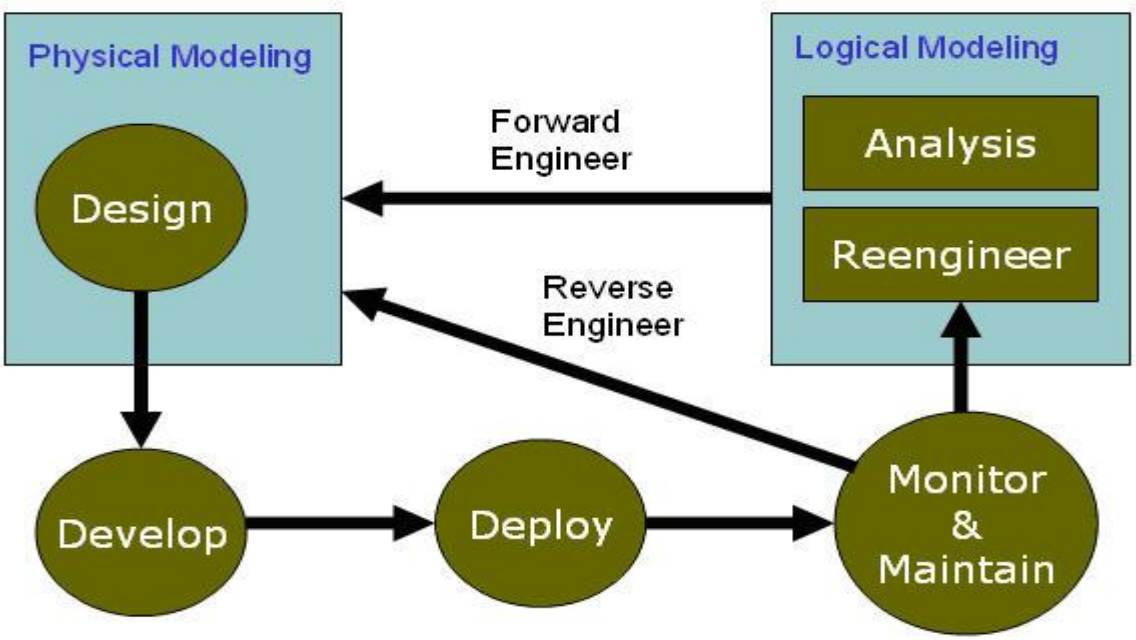


2.1.2 Reverse Engineering Defined

With society'S dependence on the Internet, many businesses need to modify their current applications, to make them web-based and move towards an electronic way of doing business. This trend has created more of an interest in code maintenance and

evolution than in the past .Thus, there is now a need for experts in older systems, as software maintenance and evolution is becoming more necessary.

Roughly, one third of total life-cycle costs are used for the programmer to understand the functionality of the existing code .Even though it is a timely and costly process, understanding the code is critical and significant, in order for a programmer to correctly make the desired changes.



Software maintenance and evolution continue to become more important as time marches on.

Reverse engineering is the act of recognizing systems elements, along with their corresponding dependencies, to generate a variety of application abstractions and design data from these system elements .To successfully do this with software, the application's code, documentation, and behavior must be studied to identify the system abstractions and various design patterns, as well as to fully understand the functionality of the system.

Software reverse engineering may be viewed as a "solution looking for a problem."

. Many programmers attempt to understand "how the code gets where it's going" and "why the code is doing something" in their everyday tasks. While there are many different approaches and techniques to reverse engineer software, their common goal is to gather as much information as possible from the current system, to assist in the maintenance task(s) at hand. This information is critical to support current maintenance and/or future development, as well as providing data to projectmanagement for planning the use of software engineering resources.

2.1.3 History of Reverse Engineering

The need for reengineering legacy systems was apparent by the early 1990s. However, with the recent pressure for businesses to go electronic, by way of the Internet, to and convert many existing systems to web-based applications, this need has intensified. There is now a demand for various methods, tools, and infrastructures to assist in transforming existing applications rather quickly and relatively inexpensively .Over the past decade, researchers have made tremendous advances in this area.
The 1980s were focused on various program comprehension theories, along with identifying the concept of reverse engineering with the evolution of software. It was noted that a majority of the software evolution process is used up by program comprehension. The topic continued to be researched throughout the 1990s. It was during this time that various infrastructures and tools were developed to assist with the main parts of reverse engineering a system .As long as an application is used, it will continuously change. As it changes, it will become more and more complex.

2.1.4 Problems with Reverse Engineering

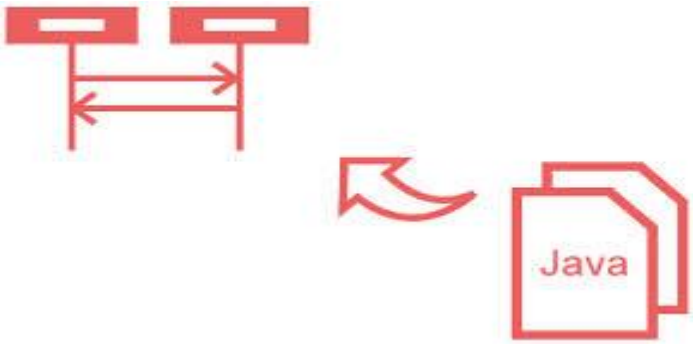
Software reverse engineering is difficult for many reasons. One reason is there might not be any documentation in the code to be modified. In some instances, the code may be complex, making the original author's purpose difficult for the new engineer to understand. Another issue occurs when the original code does not provide the correct solution for the problem. The code may have also been altered from additional problems found, creating a very cluttered and disorganized environment with which to work. The programming language may have been updated, causing new problems in the code. The software could have come from a different environment or the hardware platform may have been modified. These are just a few of the problems software engineers may face while trying to maintain code .If software engineers do not fully understand the code they are modifying, this can create future problems.

2.1.5 Importance of Reverse Engineering

Approximately $30 billion is spent a year on software maintenance, including legacy systems .An important and poor trait of legacy systems is many times business rules are intertwined within the application logic. As software lives, it is updated due to enhancements in the functionality, correction of errors, or improvements in quality. However, as software changes, the documentation is not always updated, as well. Therefore, the code becomes the

only dependable source of information when trying to understand the application's functionality. Previous design, if available, does not always map to the current implementation. Yet, effective maintenance requires a reasonably thorough understanding of the code and its intended functionality. This has led to the need for reverse engineering or some mechanism to recapture some of the original design intentions. By reverse engineering an application's code a user can then recognize the artifacts , detect various relationships, and produce abstractions that can be used to re-document and depict the initial design.

2.1.6 Practicality of Reverse Engineering



When maintaining old code, the organization will eventually need to decide if it is most cost effective to keep maintaining the existing code or if the organization should reengineer the system. There are many factors used when determining if system reengineering is appropriate. A system should be reengineered if there are regular failures, code that is out of date (about seven-to-ten years old), using application logic or structure that is excessively complicated, or code written for hardware that is obsolete. Other factors to consider for reengineering are when there is code with exceedingly large modules, unnecessary resource usage, aspects in the code that are hard-coded, difficulty in keeping resources to maintain the code, documentation that is lacking and leaves much to be desired, or unfinished design specifications. By reengineering a system, the maintainability will be improved, migration to a new environment is easier, the system tends to be more reliable, and the code is more prepared for functional enhancements.

Another reason to want to have a thorough understanding of code is the size of many applications. As they increase in size and become more complex, it becomes more important to understand their structure and behavior. Reverse engineering the code will help bring that knowledge to the user. Often, there is little information or

rationale documented behind the implementation decisions. Reverse engineering is, therefore, sometimes vital to understand the reason and logic behind existing code.

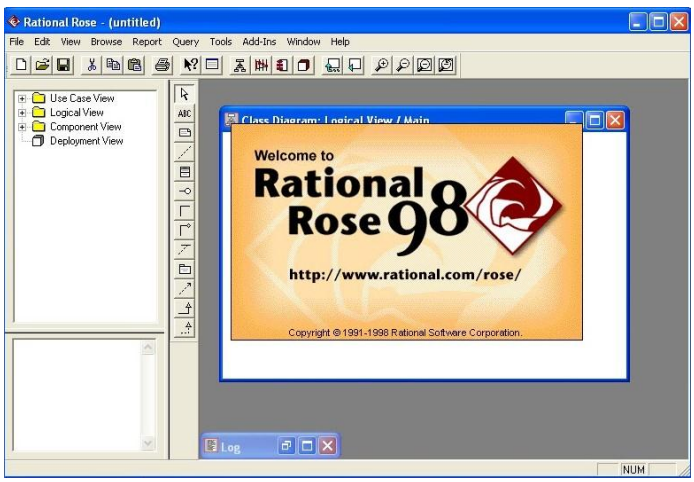
2.2 Reverse Engineering Tools

Most reverse engineering tools available, including Rational Rose®, NetBeans®, and Eclipse®, will generate a UML class diagram from Java source code. jGRASP® goes somewhat deeper by generating the class diagram, a Control Structure Diagram (CSD) which is an algorithmic level diagram, and a Viewer which will display dynamic visualizations of objects and primitives. Eclipse® provides for some additional reverse engineering functionality within the environment itself.

The tools mentioned are the more popular reverse engineering tools in common use. However, there are many others tools, including the Sun Java Studio Enterprise 7® or JBuilder®, to name a few, that will perform various software reverse engineering functions, as well. These tools will execute a variety of tasks, in addition to some of the same operations as the other tools. However, all the tools excel in different ways and possess different levels of capabilities, some are just more widely used then others.

2.2.1 Rational Rose

Rational Rose® is a modeling tool, released by the Rational Software Corporation (recently purchased by IBM), that supports, among a host of additional features, the UML graphical notation. Rational Rose® will automatically generate a UML class diagram from objectoriented source code, such as Java and C++. This is a good tool for round trip engineering, as it will allow you to create UML class diagrams from existing code, modify them, and update the source code immediately inside the application. However, there is still a good deal of human interaction required during this process.



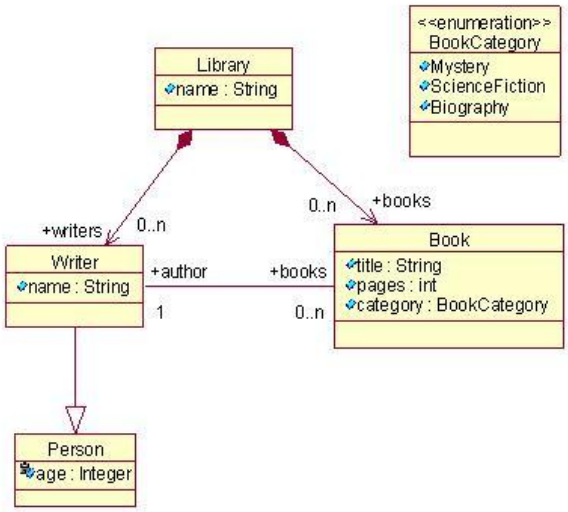
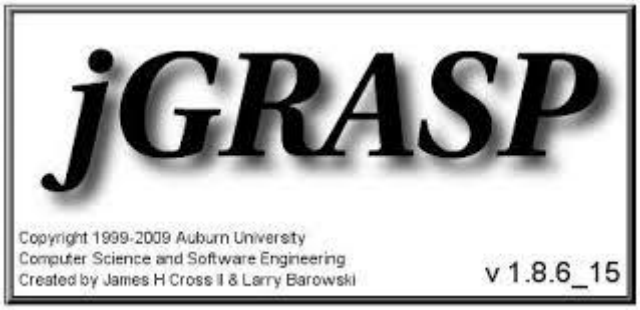


Figure 1: Rational Rose UML Class Diagram(Source: Internet)

2.2.2 jGRASP



jGRASP® is developed from pcGRASP. jGRASP® is one of the most recent applications from the GRASP (Graphical Representations of Algorithms, Structures, and Processes) .The application jGRASP® is a "lightweight integrated development environment, created specifically to provide visualizations for improving the comprehensibility of software.

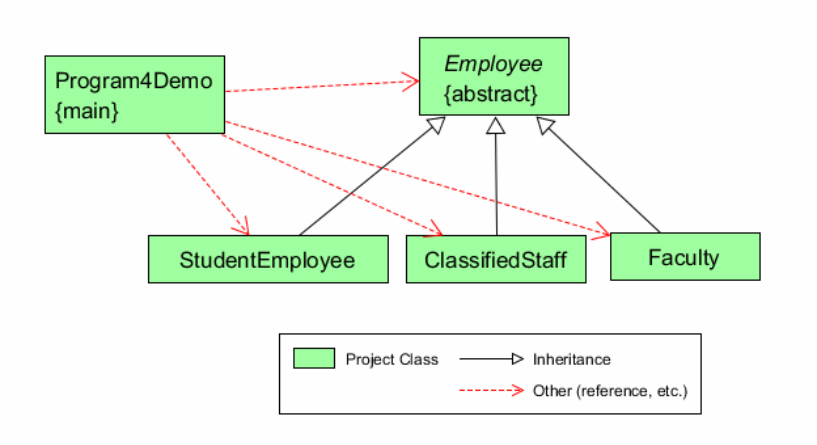


Figure 2: jGrasp CSD Diagram(Source: Internet)

jGRASP® is written in Java and supports the Java programming language, as well as C, *C++*, and Ada. jGRASP's current functionality includes the automatic generation of CSDs, UML class diagrams, and Viewers. jGRASP® alsocontains an Object Workbench and Debugger, which help a programmer to generateand debug source code.

The CSD is an "algorithmic level diagram generated for Ada, C, C++, and VHDL"

. This diagram assists the user in understanding the source code more thoroughly and in an easier manner. It will do this by representing control constructs, control paths, and the general structure of each program segment. This diagram is illustrated in the margins and indentations of the source code. This diagram is often used in the place of flow charts and other graphical diagrams. The main purpose of the CSD diagram is to "create an intuitive and compact graphical notation that is easy to use" .

jGRASP® will also generate the UML class diagram, for the Java source code from the Java class files and .jar files of a project. The diagram will illustrate the dependencies among various classes by standard UML dependency arrows. If the user selects a class, its members are displayed. If the user selects an arrow, the dependencies between the two classes are illustrated.

jGRASP® will also generate Viewers for Java source code.The Viewers, "for objects and primitives provide dynamic visualizations as the user steps through a program in debug mode or invokes a method for an object on the workbench" .Presentation views are presented for instances of classes that symbolize data structures, such as a link list, binary trees, and array wrappers. When the user opens a viewer, a structure identifier recognizes the data structure during the debugging process and displays the correct presentation view of the object for the

user.

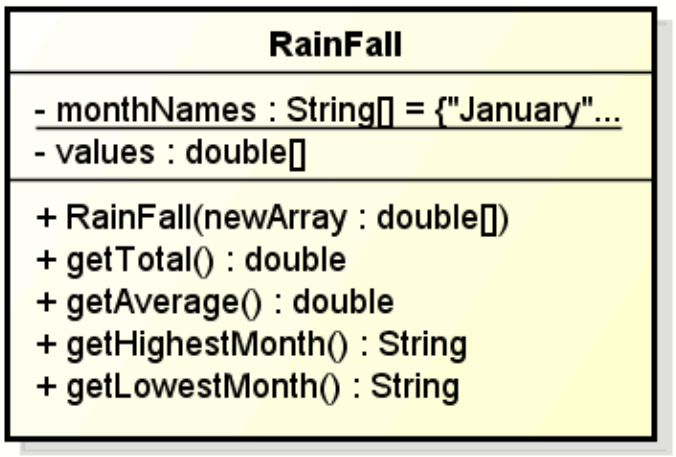


Figure 3: jGrasp UML Diagram(Source: Internet)

2.2.3 NetBeans

The NetBeans® Integrated Development Environment (IDE) is an open source application for the development and maintenance of Java application code NetBeans® will create an UML class diagram from object-oriented source code, such as Java and c++ .This tool will allow a software engineer to create UML class diagrams from existing code. The class diagram will allow the user to see potential object dependencies, thus, helping the user understand the code. However, high level, graphical object dependencies only provide limited insight to the developer. More information is vital to foster a firm grasp of what exactly is going on throughout the application logic.

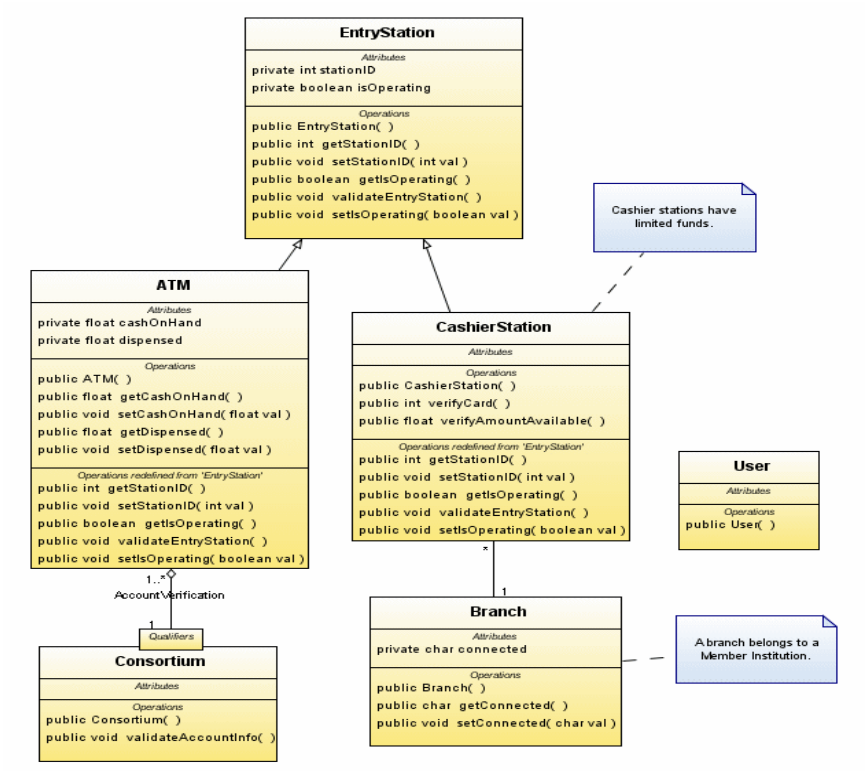


Figure 4 : NetBeans UML Diagrams (Source: Internet)

2.2.4 Eclipse

Eclipse®, another product of IBM, is an open source tool that provides an advanced development environment for various applications .Eclipse® will allow a software engineer various reverse engineering techniques while in the Eclipse® workspace. The Smart Development Environment (SDE) p1ugin for Eclipse® provides reverse engineering of Java code into UML class diagrams and output in a PDF or HTML format, entirely within the Eclipse® environment.In addition to these facilities,Eclipse® also provides for various functiona1ities within the workspace to assist in understanding program code. The Eclipse® Java IDE may assist the user by providing search capabilities for finding referenced code declarations and usages. It provides various tools for

this purpose, including Open Declaration, References, Declarations, etc. The Open Declaration operation will open a class to the selected method. The References tool will show all the references in the project for that's pecific method. The Declarations utility will show the class in which that denoted method is declared. These features may be very helpful, but it is necessary for the user to be within the project; that is, looking at the source code. There is not a way to find method dependencies up front or without being "inside" the program code.

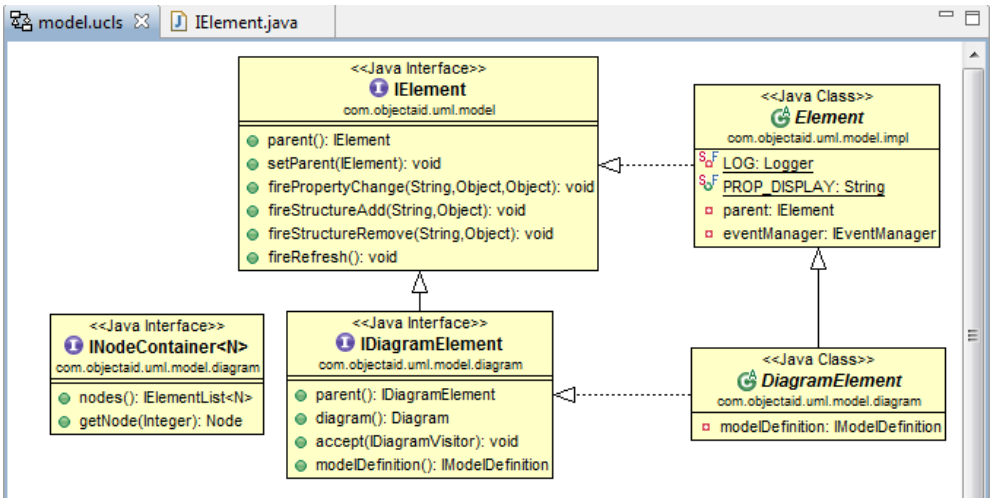
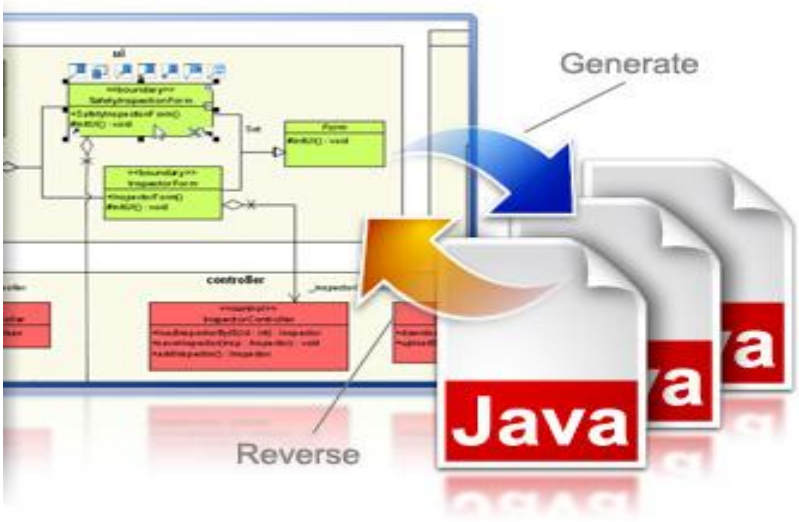


Figure 5 : Eclipse UML Diagram(Source: Internet)

CHAPTER 3 : METHODOLOGY

PROPOSED WORK : As a developer, it would be more beneficial to have a framework that drills down a Level further than providing high-level class dependencies. A comprehensive reverseengineering framework that, when given an unknown Java program, will analyze the existing structural characteristics and generate detailed low-level dependencies and relationships among code segments would be helpful in a workplace environment. The framework would, by class, show all methods declared in the class and what methods they invoke. It would also, by each method, show the class and methods it is referenced by. Equivalently, "who" invokes the services of this class and what services of other classes does "this" class invoke would be shown



3.1 Method Level Dependency Framework

This project include examining various reverse engineering tools, such as those found in Rational Rose®, Eclipse®, NetBeans®, and jGRASP®, followed by comparing and analyzing the their outputs and methodologies. Once these tools were evaluated, a new framework was developed that, when used in combination with an existing tool, will generate the UML class diagram, which is more beneficial during reverse engineering due to its focus on method level detaiL This new reverse engineering framework included accepting Java programs as input and determining the structural characteristics of the program. It provides for both a forward and reverse analyses of method level dependencies. The framework

provides two output diagrams: a complete listing by method of all classes and methods that reference the method in question, as well as an additional listing of all references made by each method in each class. While this is viewed by many as an arduous undertaking, the availability of such a framework, when used along with existing reverse engineering tools, should be helpful to the software maintenance worker.

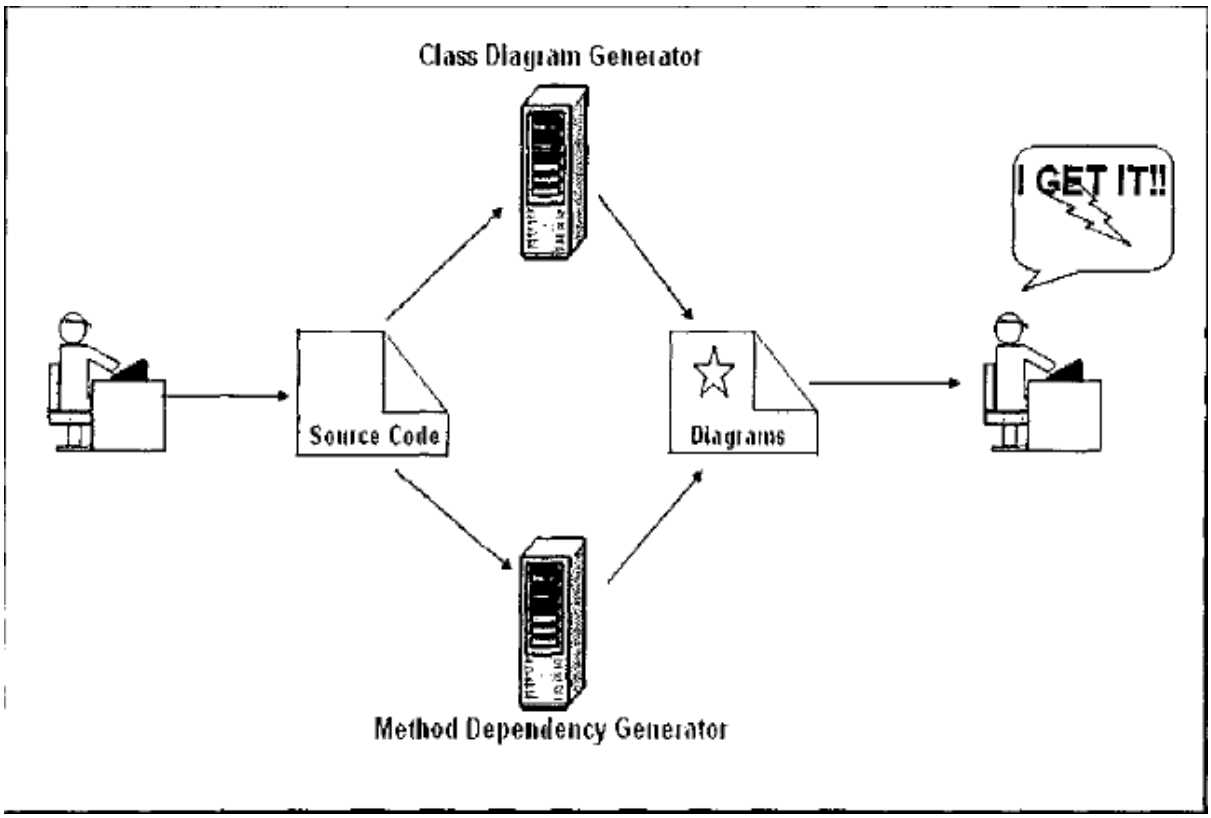


Figure 6 : UML Generator Methodology

3.2 Reverse Engineering Framework

3.2.1 Development Software

Rational Rose®, jGRASP®, NetBeans®, and Eclipse® were used to generate the various models to support reverse engineering methodologies. The method level dependency framework was developed in Java 5.0 using the Eclipse® IDE. A MySQL® database was used for storage and retrieval of various information artifacts as needed. A machine containing the Java Run-Time Environment (JRE) was utilized to run the application. This is a stand alone application and runs locally on a machine.

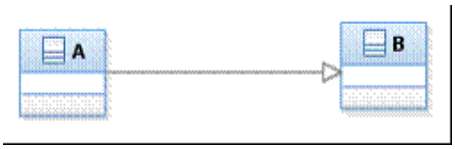
TOOLS ANS SOFTWARE USED :

- Code to be developed in JAVA
- Eclipse SDK
- Java Runtime Environment
- Windows XP/7
- Swing toolkit
- Simple Text editor

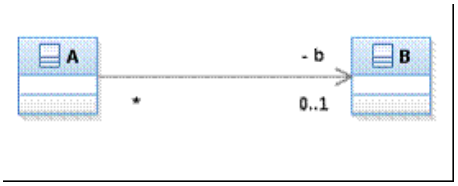
RET(Reverse Engineering Algorithm) :

Steps to reverse engineering Java program to UML class diagram :

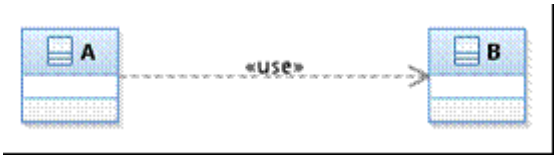
1) Draw each class or interface of the Java program in the UML class diagram with class/interface names only.
2) Identify & draw generalization (inheritance/extend) relationship between two classes in the UML class diagram, by looking for the following code pattern in the Java program:
   class A extends class B { …. }



3) Identify and draw directed association relationship between two classes in theUML class diagram, by class A { private B b; ….}



4) Identify and draw usage dependency relationship between two classes in the UML class diagram, by looking for the following code pattern in the Java program:
   class A { … method(B b) { … }



5) Class declarations were recognized by the keyword "class" and a space.

6) Class instantiations were distinguished by the keyword "new" followed by a space, a word (characters a-z, A-Z, 0-9), and then a left parenthesis.

7) Method declarations were identified by a word.

8) The method calls were discovered by checking for a word and checking to see if they have not already been labelled a class instantiation

3.2.2 Framework Development

3.2.2.1 Framework Design

The project component was developed using the Java programming language and was organized in a modular format. It consisted of fiveclasses: MainFrame.java, FileHandler.java, DatabaseMethods.java, GenerateDiagrams.java, and Constants.java. Each class, composed of various methods, was designed to handle a different part of the application functionality.

From here, the various class dependencies can be seen, along with the global variables and methods found in each class.

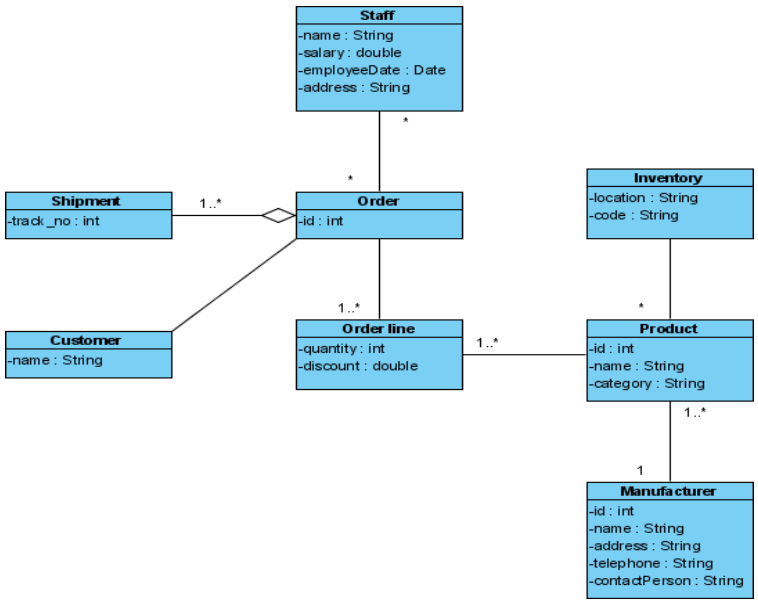


Figure 7 : Example of class Diagram

3.2.2.1.1 MainFrame.java

MainFrame.java was designed to generate and handle the Graphical User Interface that runs as a stand alone application for this framework. The GUI was developed using the Swing toolkit in Java, which is part of the Java Foundation Classes. This toolkit allowed for easy use of standard components, such as textboxes, panels, buttons, frames, etc. This class contained the main method. It built the GUI and controlled any action taken within the GUI by calling the corresponding methods to accomplish that task. The methods found in Mainframe.java are listed in Figure .

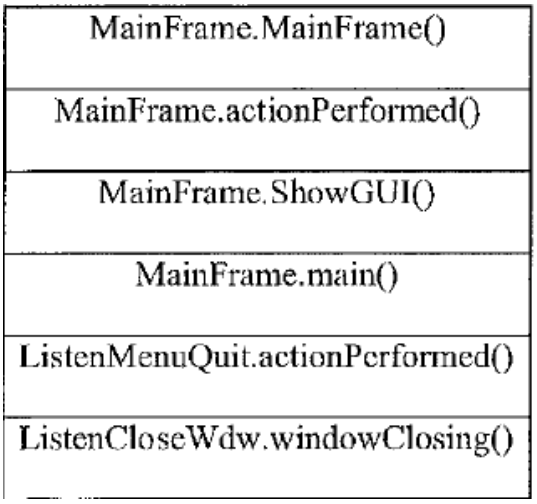


Figure 8 : MainFrame.java Method List

3.2.2.1.2 FileHandler.java

FileHandler.java managed and evaluated the data coming in through the input files. This class contained methods that read in the Java source code and, considering the order, examined it for the structural characteristics that would indicate a class or a method. First the entire input was read and any leading or trailing spaces and line feeds were removed, storing the input as a StringBuffer. Next, the input StringBuffer was parsed by open brackets, close brackets, or semi-colons, until the entire file was read. Each substring was evaluated to determine if it contained a class declaration, an object instantiation, a method declaration, or a method call. This was accomplished in Java through the use of regular expressions, also known as patterns. Regular Expressions were created to recognize the class declarations, method declarations, all class instantiations, and any method calls in each given file.

• Class declarations were recognized by the keyword "class" and a space.
• Class instantiations were distinguished by the keyword "new" followed by a space, a word (characters a-z, A-Z, 0-9), and then a left parenthesis. The referenced name and the class name are both stored in a hash map for matching later.
• Method declarations were identified by a word (characters a-z, A-Z, 0-9) followed by a space, a word (characters a-z, A-Z, 0-9), and ending with a left parenthesis.
• The method calls were discovered by checking for a word (characters a-z, A-Z, 0-9) followed by a left parenthesis and checking to see if they have not already been labeled a class instantiation.

If part of the string matched a pattern, it was then checked for reserve words and parsed out by the characters, to get the actual class or method name. If a method call was found, it was stored in the database. The methods created to manipulate the input files are listed in Figure .

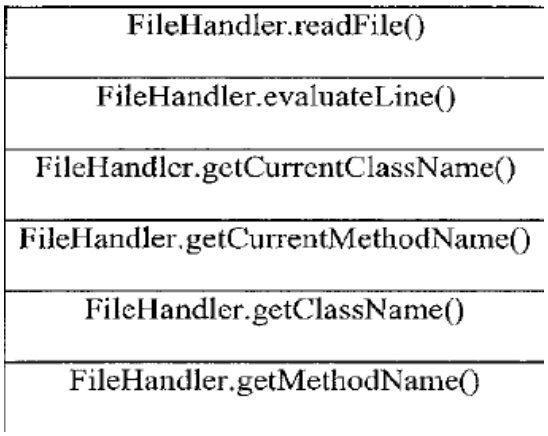


Figure 9 : FileHandler.java Method List

3.2.2.1.3 DatabaseMethods.java

DatabaseMethods.java manipulated the database. This class made the connection to the MySQL® database. It was also responsible for any calls to update or query the database throughout the framework. As method calls were identified by the FileHandler class, they were saved to the database. As the user selected to generate diagrams from the Mainframe class, this information was retrieved from the database. Figure 11 contains all the methods found in DatabaseMethods.java

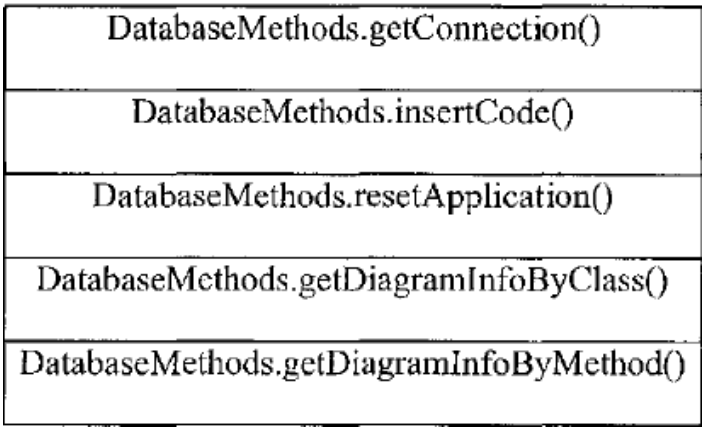


Figure 10 : DatabaseMethods.java Method List

3.2.2.1.4 GenerateDiagrams.java

GenerateDiagrams.java was used to generate the output diagrams. The user has the ability to generate two different diagrams

The methods located in GenerateDiagrams.java are listed in Figure

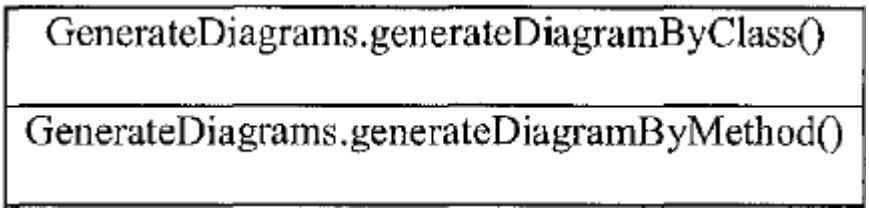


Figure 11 : GenerateDiagrams.java Method list

3.2.2.1.5 Constants.java

Constants.java defined all constants used throughout the framework. The UML Generator used constants to define the various patterns it was searching for, in each class and image locations. The constants used in the framework are provided in Figure

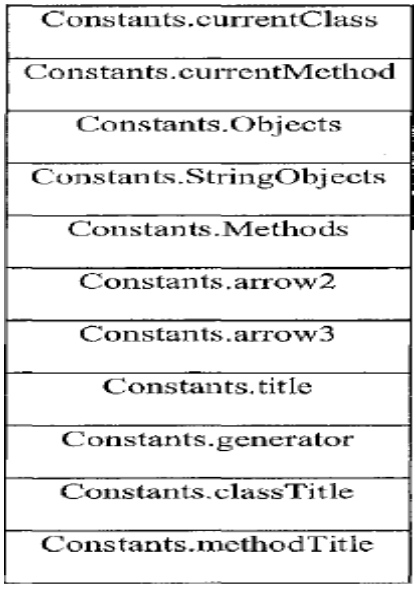


Figure 12 : Constants.java Constants list

3.2.2.2 Database Design

The MySQL® database created to store the infonnation was named "thesis." The thesis database only contained one table, "code." This table consisted of four columns, CurrentClass, CurrentMethod, CalledClass, and CalledMethod. While scanning the input file, as a method call was found in a class, the current class, current method, called class, and called method were stored in the code table. This table was queried, in order to generate the diagrams. The database diagram is found in Figure.

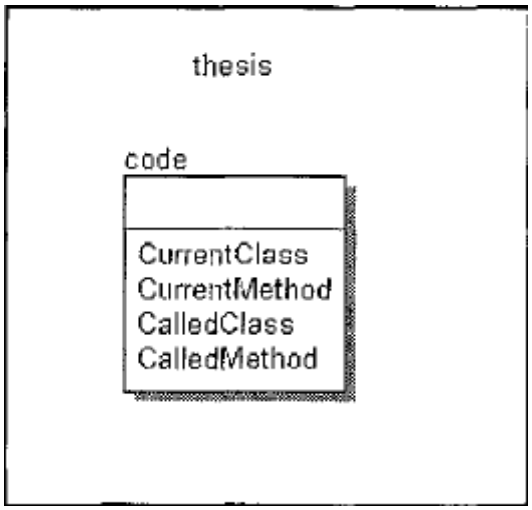


Figure 13 : Database Diagram

3.2.3 Framework Functionality

The reverse engineering framework was relatively simple to operate and assumed the input java files would compile together. The user began by starting the application. They were given a graphical user interface that would allow them to manipulate the framework. This GUI is shown in Figure .

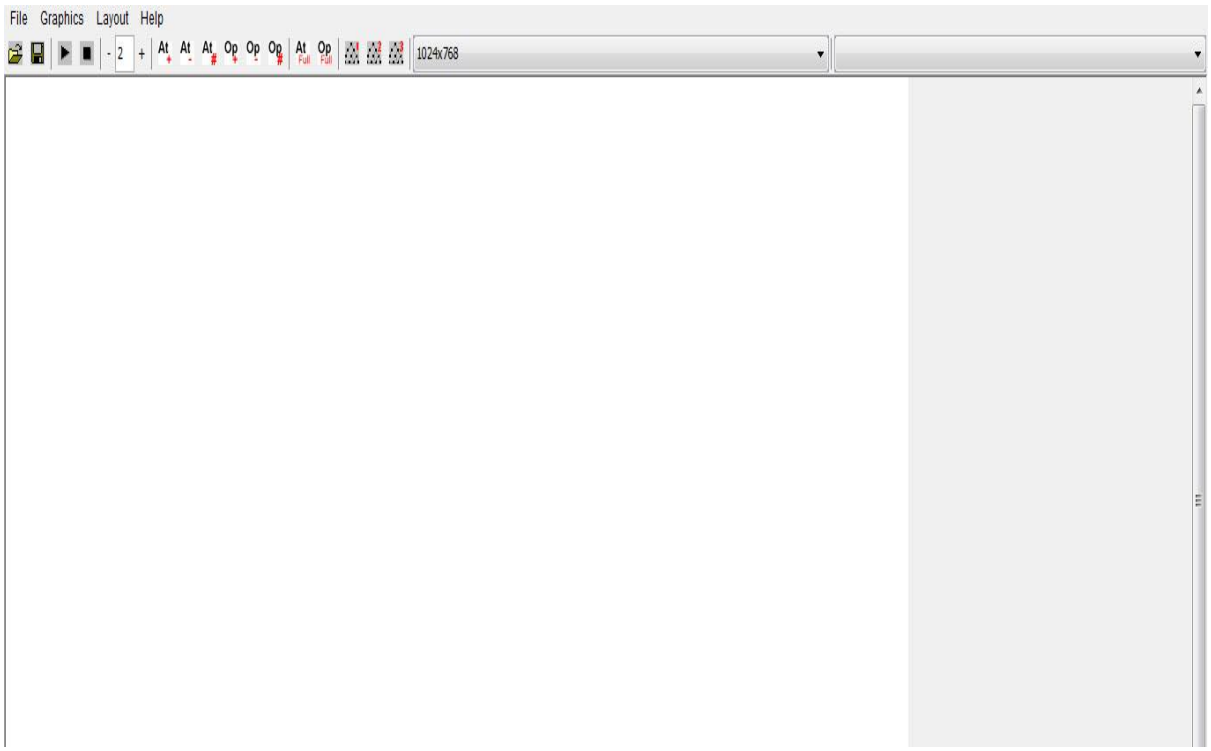


Figure 14: Framework Diagram

The user selected one file at a time to examine, by selecting the "Open File to Read" button and choosing the file. A file selector would appear and the user had to browse to find the desired file, as shown in Figure .

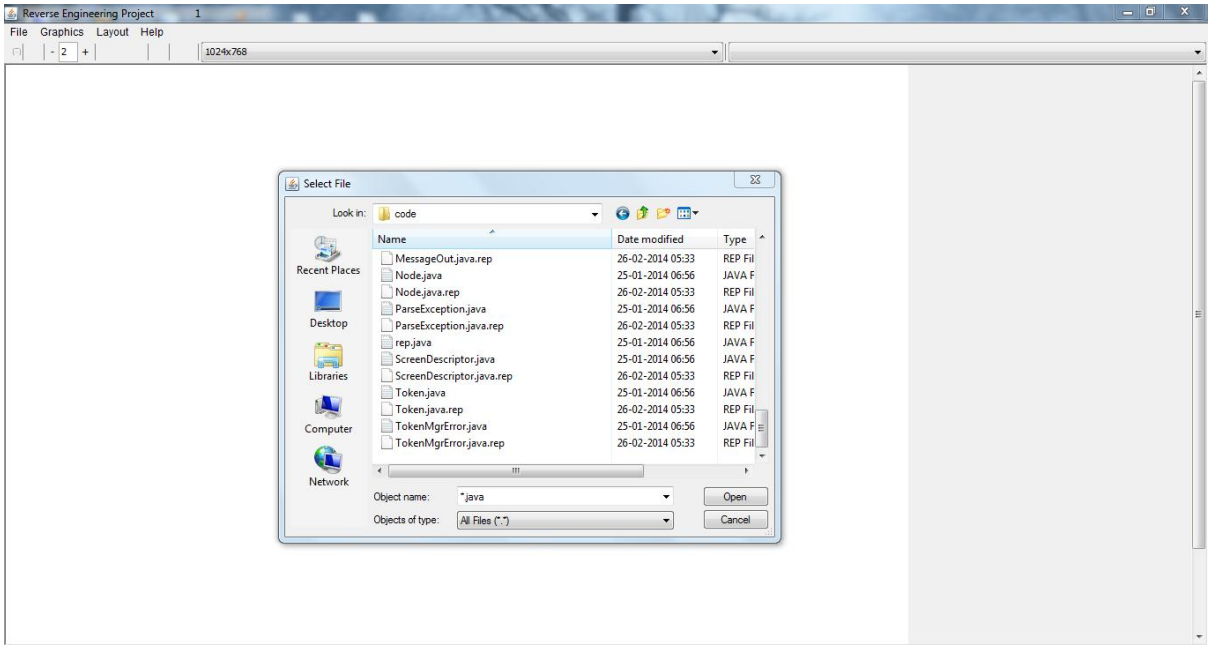


Figure 14: Framework file open diagram

The file was uploaded and scanned for the various structural characteristics, indicating a class declaration, method declaration, class instantiation, or a method call. As a method call was found, the current class, current method, the class in which the called method was contained, and the called method name were all saved to the database. This was repeated for each file the user wished to read. The user was able to view a list of all files read, thus far, in the file list in the GUI. In order to quit the application, the user can select File on the menu bar and the Quit option. The Help option on the menu bar would be used to provide the user with help information. The Method Level interface can be seen in Figure

The information can be cleared from the database by selecting the "Reset Application" button, in order to start clean again.

3.3 Framework Output

There are diagrams that were generated by the UML Generator. These diagrams included a diagram by class showing the class and method calls from it, illustrated in Figure .

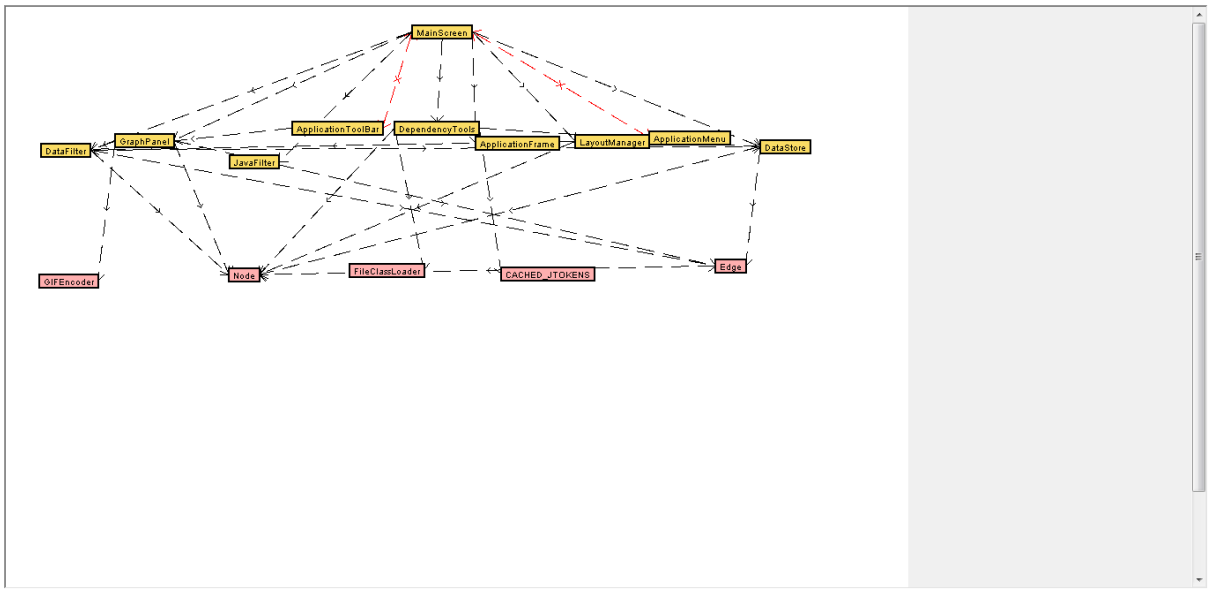


Figure 15: Framework output

The diagram (Figure) shows a representation of all classes, the methods that are in them, and what methods they depend on. From this diagram, the user was able to see by class what other classes and methods a change would potentially affect.

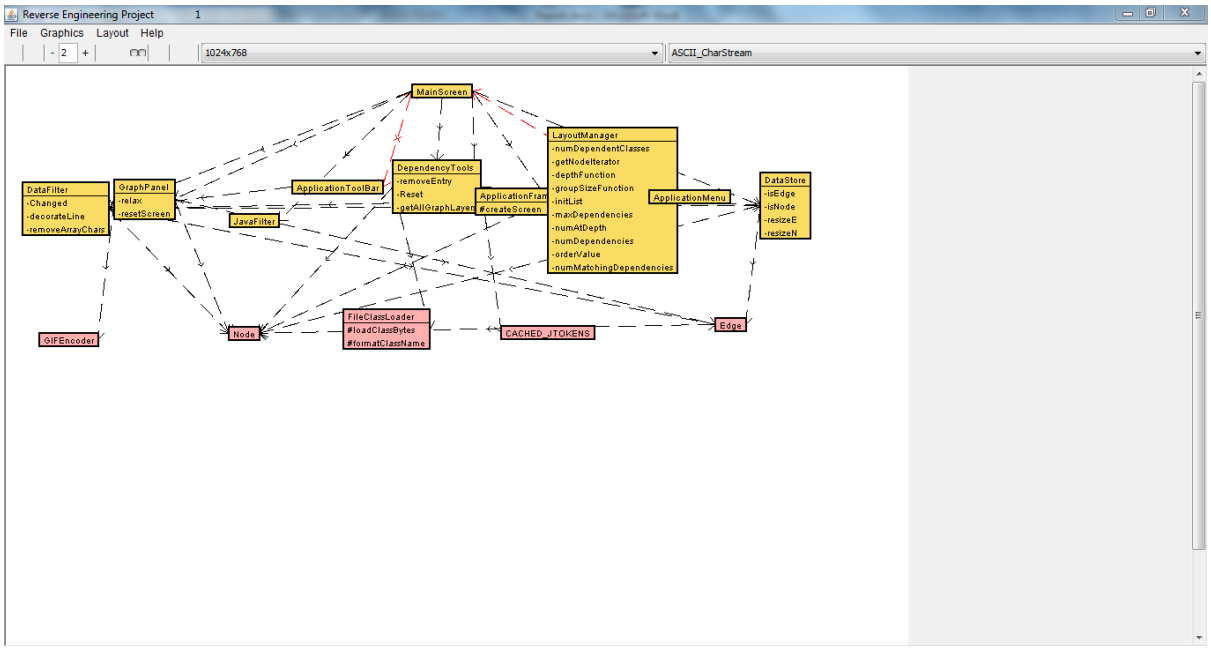
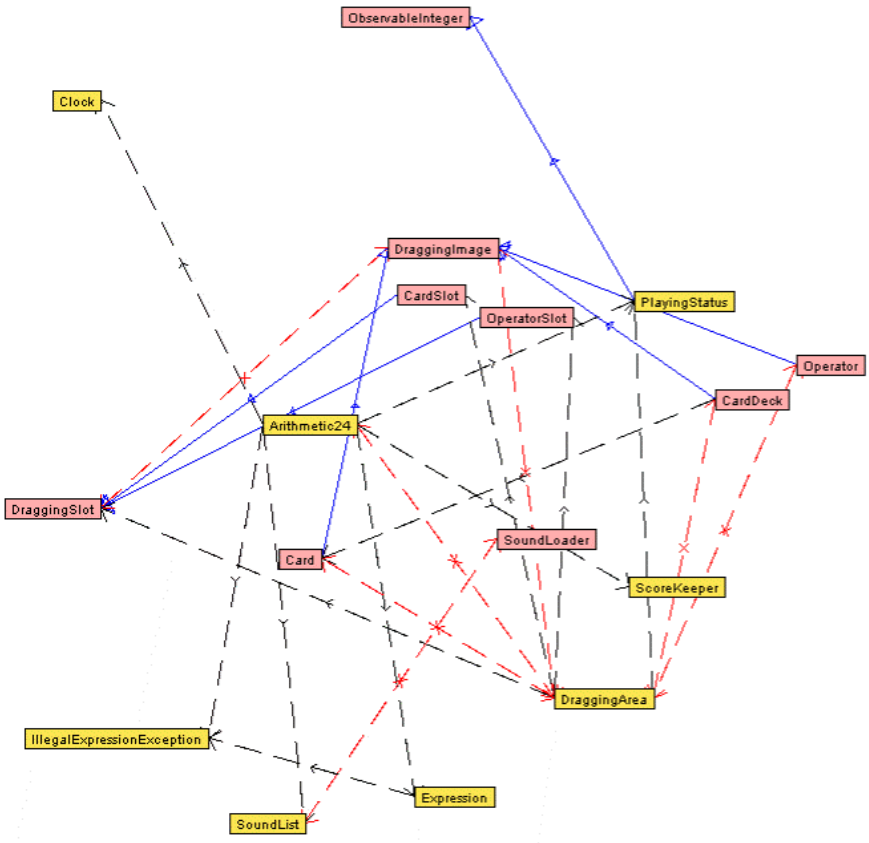


Figure 16: Framework showing methods and classes

.Upon completion of this new reverse engineering scheme, both method level diagrams were compared with the existing diagrams generated by the other commonly-used approaches. This was accomplished by analyzing the results for each diagram. By examining output from the existing methods, along with output provided by this new reverse engineering framework, it was apparent that the new framework provided a greater level of detail. The provision of method level dependencies in combination with the output of existing tools should provide a more practical tool for software maintenance.

CHAPTER 4 : RESULTS

The test cases for the new reverse engineering framework provided for the display of method level Dependencies , in addition to the diagrams of existing tools. The UML Generator was designed to read in Java source code input files as a source for the generation of the desired detailed diagrams. The test bed for this thesis contained many different test cases obtained from various sources. Each test case was made up of multiple class files, all varying in different characteristics, such as size and functionality. All test files contained, at the minimum, the essential information to retrieve the desired results, such as method calls and the respective method signatures. The important factor for this thesis was to present the method dependencies; thus, the test files focused on method calls. In order to test the new methodology, each test case was compiled and loaded into the reengineering framework. This generated the diagrams to display the lower level dependencies in the questionable application. The new dependency approach is beneficial when used in conjunction with existing software that shows high level dependencies. Many test cases were run and the results attest to this finding. **One example of the test case is as shown below which takes input of several classes and present a class diagram showing dependencies**.



CHAPTER 5 : OBSERVATIONS AND CONCLUSION

Results clearly indicate the comprehensive nature of the framework that includes, not only useful UML class diagrams, but the essential addition of method level dependencies. By viewing these results it is clear the new framework, consisting of detailed method level dependencies in conjunction with higher level class diagrams, is a useful methodology for undertaking real world software maintenance. As each test case was evaluated, the framework was found to reliably produce lower level dependencies among complex Java methods. Diagrams produced within the framework provide a quick visual artifact of method level detail within specific applications. The reliability of maintenance activities should be much improved by the use of this framework in the workplace.

Each of the test cases demonstrated the use of the new framework that provides the software practitioner with a view of the source code characterized by a lower level of granularity. By examining the results for each scenario, the developer may readily observe the results of using this new framework. The results demonstrate how the UML class diagram provided for high level, architectural information of the application. However, this information alone leaves much to be desired regarding application specific logic and detail, which is where most code changes (and errors) occur. Both of the method level dependency diagrams assist the developer with a more detailed view of dependencies in code. In particular, the Diagram By Class facility indicates all methods each of the instance methods call, thus providing a map to other services provided by other methods in other classes. To complement the Diagram By Class facility, the Diagram By Method presented for each method in a class those methods in other classes that have dependencies upon the particular method. In summary, UML class diagrams, supplemented with diagrams by class and by method, provide a comprehensive framework to assist the software maintenance practitioner.

The reverse engineering framework has the potential to expedite many of the activities of those engaged in software development. This new method dependency approach provides a very practical, lower level of granularity that should be useful to professionals in the

workplace. By coupling this new approach with existing software that generates UML diagrams with their higher level architectural descriptions of collaborating classes, the practitioner now posses a comprehensive methodology that addresses both the architectural class dependencies and other class relationships, along with a more detailed analysis of application design and code central to modem day development and maintenance needs.
Method level detail provides a higher degree of assurance in reconciling a myriad of maintenance duties in the workplace. While UML class diagrams are very helpful in displaying class relationships, the additional detailed information provided by this method level generator completes a comprehensive strategy, which should provide for significantly improved software maintenance efforts.

5.2 Future Work

There are several opportunities for future work, which may extend the utility of this framework and provide additional workplace value to software engineering practitioners.

The UML Generator was developed using Java. This framework recognized various ways to declare new classes, new methods, and method calls throughout various files loaded into the system. However, the Java programming language is quite complex, therefore, some potential enhancements exist.
The framework could be modified to be more robust and handle the entire range of Java syntax, such as recognizing every way an object can be instantiated. Some known issues, not yet accounted for in the UML Generator, include the capability to recognize creating an object and instantiating it separately and to recognize multiple functions within a line, such as declaring an object within a method call.
Similarly, the framework is not set up to account for implementing interfaces in Java. This is because a class can implement multiple interfaces. With the current design of the generator, when the application encounters a method call, it will not know for certain if the method is found in the current file or in one of the interfaces.
The new approach could also be improved by integrating this generator in with the existing technology for developing a class diagrams, such as Rational Rose®, Eclipse®, or jGRASP®. This could be accomplished various ways, such as, when the user views the class diagram, a provision could be made to click on a class to view the method dependencies.

REFERENCES:

Ali, Muhammad Raza, "Why Teach Reverse Engineering," ACM SIGSOFT Software Engineering Notes, Volume 30, Issue 4; pp. 1-4, July *200S.*

Buss, Erich and John Henshaw, "A Software Reverse Engineering Experience," Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative Research CASCON '91, pp. *SS-72,* October 1991.

Chen, Zhixiong and Delia Mars, "Experiences with Eclipse IDE in Programming," Consortium for Computing Sciences in Colleges, pp. 104-112, *200S.*

Demeyer, Serge, Stephane Ducasse, and Oscar Nierstrasz, "Finding Refactorings via Change Metrics," ACM SIGPLAN Notices, Proceedings of the *1Sth* ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA '00, Volume *3S,* Issue 10, pp. 166-177, October 2000.

Muller, Hausi, "Reverse Engineering Strategies for Software Migration." Proceedings of the 19th international conference on Software Engineering ICSE '97, pp. 659-660, May 1997.

Newcomb, Philip, "Web-Based Business Process Reengineering," IEEE Software, pp. 116-118, November 1995.

Sneed, Harry, "Planning the Reengineering of Legacy System," IEEE Software, pp. 24-34, January 1995.

Eclipse, "Eclipse - an open development platform," Eclipse, http://www.eclipse.org/, last accessed 2007.

APPENDIX A:

SOURCE CODE : Constants.java

```
public class Constants
{ public static final String currentClass = "(class) (\ \s) ";
public static final String currentMethod = "(\\w+) (\\[\\])*(\\s) (\\w+) (\\s)*(\\()";
public static final String Objects = "(new) (\ \s) (\ \w+) (\ \s) * (\ \ ()";
public static final String StringObjects = " (String) (\ \s) (\ \w+) (\ \s) * (=)";
public static final String Methods = "(\\.)*(\\w+) (\\s)*(\\()";
public static final String arrow2 = "C:/Documents and
Settings/ankz/workspace/Thesis/images/arrow2.jpg";
public static final String arrow3 = "C:/Documents and
Settings/ankz/workspace/Thesis/images/arrow3.jpg";
public static final String title = "C:/Documents and
Settings/ankz/workspace/Thesis/images/Title.jpg";
public static final String generator = "C:/Documents and
Settings/ankzworkspace/Thesis/images/generator.jpg";
public static final String class Title = "C:/Documents and
Settings/ankz/workspace/Thesis/images/ClassTitle.jpg";
public static final String methodTitle = "C:/Documents and
Settings/ankz/workspace/Thesis/images/MethodTitle.jpg";
```

APPENDIX B

SOURCE CODE : DatabaseMethods.java

```
import java.sql.*;
import java.util.*;
public class DatabaseMethods
{
public Connection connection = null;
public void getConnection() throws SQLException
{
try
{
// Load the JDBC driver
// MySQL MM JDBC driver
String driverName = "org.gjt.mm.mysql.Driver";
Class.forName(driverName) ;
//Create a connection to the database
String serverName = "localhost";
String mydatabase = "thesis";
String urI = "jdbc:mysql://" + serverName + "/" + mydatabase;  // a JDBC urI
String username = "root";
String password = "root";
connection = DriverManager.getConnection(url, username, password) ;
}
catch (ClassNotFoundException e)
{
System.out.println("could not find the database driver") ;
catch (SQLException e)
{
System.out.println("could not connect to the database") ;
```

```
finally

{

}

public void insertCode(String CurrentClass, String CurrentMethod, String CalledClass, String CalledMethod) throws SQLException

{

PreparedStatement Stat = null;

try

{

if (connection == null)

{

getConnection();

}

String sSQLString =" INSERT INTO code (CurrentClass, CurrentMethod, CalledClass, CalledMethod) values(?,?,?,?)";

Stat = connection.prepareStatement(sSQLString);

Stat.setString(l, CurrentClass);

Stat.setString(2, CurrentMethod);

Stat.setString(3, CalledClass);

Stat.setString(4, CalledMethod);

if (Stat.executeUpdate() == 0)

{

System.out.println("did not insert");

}

}

catch (SQLException e)

{

System.out.println("SQL Exception" + e);

}

finally
```

```
{
Stat.close();
connection.close();
}
}
public void resetApplication() throws SQLException
{
PreparedStatement Stat = null;
try
{
if (connection == null)
{
getConnection();
}
String sSQLString = "delete from code";
Stat = connection.prepareStatement(sSQLString);
if (Stat.executeUpdate() == 0)
{
System.out.println("did not clear out the
database") ;
}
}
catch (SQLException e)
{
System.out.println("SQL Exception: " + e);
}
finally
{
Stat.close();
```

```
connection.close();
}
}
public HashMap getDiagramlnfoByClass() throws SQLException
{
PreparedStatement Stat = null;
ResultSet ResultSet = null;
HashMap Results = new HashMap();
try
{
if (connection == null)
{
getConnection();
}
String sSQLString = "SELECT distinct * FROM code order by CurrentClass, CurrentMethod;";
Stat = connection.prepareStatement(sSQLString);
ResultSet = Stat.executeQuery();

int counter = 1;
while (ResultSet.next())
{
Results.put("CurrentClass" + counter, ResultSet.getString ("CurrentClass " ) ) ;
Results.put("CurrentMethod" + counter,ResultSet.getString("CurrentMethod")) ;
Results.put("CalledClass" + counter, ResultSet.getString ("CalledClass") ) ;
Results.put("CalledMethod" +counter,ResultSet.getString("CalledMethod") ) ;
counter++;
}
}
catch (SQLException e)
```

```
{
System.out.println("SQL Exception" + e);
}
finally
(
Stat.close();
ResultSet.close();
connection.close();
}
return Results;
}
public HashMap getDiagramInfoByMethod() throws SQLException
(
PreparedStatement Stat = null;
ResultSet ResultSet = null;
HashMap Results = new HashMap();
try
{
if (connection == null)
{
getConnection();
}
String sSQLString =
"SELECT distinct * FROM code order by CalledClass, CalledMethod ";
Stat = connection.prepareStatement(sSQLString);
ResultSet = Stat.executeQuery();
int counter = 1;
while (ResultSet.next())
{
```

```
Results.put("CurrentClass" +
counter,ResultSet.getString("CurrentClass") ) ;
Results.put("CurrentMethod" + counter,ResultSet.getString("CurrentMethod"));
Results.put("CalledClass" +counter,ResultSet.getString("CalledClass") ) ;
Results.put("CalledMethod" + counter,ResultSet.getString("CalledMethod") ) ;
counter++;
}
}
catch (SQLException e)
{
System.out.println("SQL Exception" + e);
}
finally
{
Stat.close();
ResultSet.close();
connection.close();
}
return Results;
}
}
```

APPENDIX C

SOURCE CODE : FileHandler.java

```
import java.io.*;
import java.sql.*;
import java.util.regex.*;
import java.util.*;
public class FileHandler
{
int bracketCounter = 0;
String CurrentClassName = "";
String CurrentNestedClass = "";
String CurrentMethodName = "";
String ExtendedClass = "";
String NestedExtendedClass ="";
HashMap ObjectList = new HashMap();
HashMap DeclaredMethods = new HashMap();
HashMap CalledMethods = new HashMap();
public void readFile(File file) throws IOException,
SQLException
{
try
{
FilelnputStream fis = new FilelnputStream(file);
BufferedlnputStream bis = new
BufferedlnputStream(fis);
DatalnputStream dis = new DatalnputStream(bis) ;
String sText = "";
StringBuffer sResult = new StringBuffer("");
while ((sText= dis. readLine ()) ! = null)
```

```
{
sText=sText.replaceAll("A\\s+", "");
sText=sText.replaceAll("\\s+$","");
sText=" " + sText;
sResult. append (sText) ;
}
int index1= 0;
int index2 =0;
int index3 =0;
StringBuffer sTemp = new StringBuffer("");
while (sResult.length() != 0)
{
index 1 = sResult.indexOf("{");
index2 = sResult.indexOf(";");
index3 = sResult.indexOf("}");
if (index3 == 0)
{
sTemp.replace(0, sTemp.length(), sResult.substring(0,index3+1));
if (sResult.length() > 2)
    sResult.replace(0,
    sResult.length(),
    sResult.substring(ind
    ex3+2,
    sResult.length()));
else
sResult.replace(0,sResult.1ength (), "");
}
else if ((index1 < index2) && (index1 != -1))
{
```

```
sTemp.replace(0, sTemp.length(), sResult.substring(0, indexl+l));
sResult.replace(0, sResult.length(), sResult. substring (index1+2, sResult.length()));
}
else if ((index2 < index3) && (index2 != -1))
{
sTemp.replace (0, sTemp.length(),sResult.substring( 0, index2+1));
sResult.replace(0, sResult.length(), sResult.substring(index2+2, sResult.length()));
}
else
{
sTemp.replace(0, sTemp.length(), sResult.substring(O, index3+1));
if (sResult.length() > 2)
sResult.replace(0, sResult.length() , sResult. substring (index3+2, sResult.length()));
else
sResult.replace(0,sResult.length(), " ") ;
evaluateLine(sTemp);
}
//go through the CalledMethods and see if they are
//declared in the Classes read ...
int Dcounter =  DeclaredMethods.size()/2;
int Ccounter = CalledMethods.size()/4;

String CalledClass = "";
String CalledMethod = "";
String DeclaredClass = "";
String DeclaredMethod = "";
boolean bFound = false;
for (int i = 1; i<=Ccounter; i++)
{
```

```
bFound = false;
CalledClass = (String) CalledMethods.get("Class" + i);
CalledMethod = (String) CalledMethods.get("Method" + i);
String CuMethod = (String) CalledMethods.get("CurrentMethod" + i);
for (int j = 1; j<=Dcounter; j++)
DeclaredClass = (String) DeclaredMethods.get("Class" + j);
DeclaredMethod = (String)DeclaredMethods.get("Method" +j) ;
If (CalledClass.equalslgnoreCase(DeclaredClass) & CalledMethod.equalsIgnoreCase(Dec
laredMethod) )
{
//save to the database- as in that class
 DatabaseMethods dataMethods = new DatabaseMethods();
if (CalledClass . length () ! = 0 &&
CuMethod.length() !=0 &&
CalledClass.length() !=0 &&
CalledMethod.length() !=0)
dataMethods.insertCode(Called Class.trim(), CuMethod.trim(), CalledClass.trim(),
CalledMethod.trim());
bfound=true;
}
if (bFound)
break;
}
if (!bFound)
{
//save to the database as inherrited
String XClass = (String)
CalledMethods.get("ExtendedClass" + i);
DatabaseMethods dataMethods = new DatabaseMethods();
if (CalledClass . length () ! = 0 && CuMethod. length () !=0 && XClass.length() !=0&&
```

```
CalledMethod.length() !=0)
dataMethods.insertCode(CalledClas
s.trim(), CuMethod.trim(),
XClass. trim () ,
CalledMethod.trim());
}
}
fis.close () ;
bis.close();
dis.close();
}
catch (IOException e)
{
}
}
public void evaluateLine(StringBuffer sInput) throws
SQLException
{
Pattern pattern = null;
Matcher matcher = null;
Constants myConstants = new Constants();
int inputLength = sInput.length();
for (int i = 0; i<inputLength; i++)
{
if (sInput.charAt(i) == ' { ' )
{
bracketCounter++;
if (sInput.charAt(i) == '}')
{
```

```
bracketCounter--;
}
}
if (bracketCounter == 1)
{
CurrentNestedClass ="";
NestedExtendedClass = "";
//set up the current class pattern
String currentClass = myConstants.currentClass;
pattern = Pattern.compile(currentClass);
matcher = pattern.matcher(sInput);
if (matcher.find())  //if i find a new class
{
getCurrentClassName(sInput);
return;
}
//get any new class instantiation (object)
String Objects = myConstants.Objects;
pattern = Pattern.compile(Objects);
matcher = pattern.matcher(sInput);
if (matcher.find()) //if the line contains an object
{
getClassName(sInput);
return;
}
else //look for the other String declaration
String StringObjects = myConstants.StringObjects;
pattern = Pattern.compile(StringObjects);
matcher = pattern.matcher(sInput);
```

```
if (matcher.find()) //if the line contains an object
{
getClassName(sInput);
return;
}
}
//if the bracketCounter is greater than 0- then i need to look for methods
if (bracketCounter > 0)
{
//set up the current method declaration pattern
String currentMethod = myConstants.currentMethod;
pattern = Pattern.compile(currentMethod);
matcher = pattern.matcher(sInput);
if (matcher.find()) //if i find a new method
{
getCurrentMethodName(sInput);
return;
}
//this needs to be done last and if it passes the other test before it
//get any method calls
String Methods = myConstants.Methods;
pattern = Pattern.compile(Methods);
matcher = pattern.matcher(sInput);
if (matcher.find())
{
getMethodName(sInput);
return;
}
}
```

```
}
public void getCurrentClassName(StringBuffer Line)
{
StringTokenizer st = new
StringTokenizer(Line.toString());
String Next = "";
while (st.hasMoreTokens())
{
Next = st.nextToken();
if (Next.equalsIgnoreCase("class"))
{
if (bracketCounter < 2)
CurrentClassName = st.nextToken();
else
CurrentNestedClass = st.nextToken();
}
if (Next.equalsIgnoreCase("extends"))
{
if (bracketCounter < 2)
ExtendedClass = st.nextToken();
else
NestedExtendedClass = st.nextToken();
}
}
}
public void getCurrentMethodName(StringBuffer Line)
{
StringTokenizer st = new
StringTokenizer(Line.toString());
```

```
String Next = "";
String Temp = "";
String Class = "";
//handle else if
if (Line.toString() .contains("else if"))
{
return;
}
while (st.hasMoreTokens())
{
Next = st.nextToken();
if (Next.contains(" ("))
{
int index = Next.indexOf("(");
if (index ! = 0)
Temp Next;
CurrentMethodName = Next. substring (0,
index) ;
else
CurrentMethodName Temp;
}
Temp=Next;
}
//i want to store all declared methods to a hashmap
int counter = DeclaredMethods.size()/2;
counter++;
if (CurrentNestedClass.length() >0)
Class = CurrentNestedClass;
```

```
else
Class = CurrentClassName;
DeclaredMethods.put("Class" + counter, Class);
DeclaredMethods.put("Method" + counter,
CurrentMethodName) ;
}
public void getClassName(StringBuffer Line)
{
StringTokenizer st = new
StringTokenizer(Line.toString());
String Class = st.nextToken();
Class.trim () ;
String Reference = st.nextToken();
Reference.trim();
int counter = ObjectList.size()/2;
counter++;
ObjectList.put("Class" + counter, Class);
ObjectList.put("Reference" + counter, Reference);
}
public void getMethodName(StringBuffer Line) throws
SQLException
{
String Next = "";
String Class = "";
String Reference ="";
String Method = "";
int index = 0;
int index2 = 0;
Next = Line.toString();
```

```
if (Next.contains("."))
{
index = Next.indexOf(".");
index2 = Next.indexOf("(");
int index3 =Next.indexOf("=");
int index4=Next.indexOf(":");
int index5=Next.indexOf(")");

//check for reserve words
if (index2 == -1)
{
return;
}
else if ((index2< index))
{
String subNext = Next. substring (0, index2);
if (subNext. trim (). contains (" if") || subNext.trim().contains ("catch") ||
subNext.trim().contains("do") ||subNext.trim ().contains (" for") I|| subNext. trim ().contains ("return") ||
subNext.trim() . contains ("switch") I IsubN
ext.trim() .contains("while"))
{
string inside =
Next. substring (index2+1,
Next.length());
if (inside.indexOf(".") == -1)
{
return;
}
else
```

```
{
StringBuffer sbinside = new
StringBuffer() ;
sbinside.append(inside);
getMethodName(sbinside);
return;
}
}
else
{
if (subNext.length()== 0)
{
return;
}
else
{
if (index2 < index && index5 <
index) //this is for casting
{
String inside=Next.substring(index5
+1
, Next.length());
StringBuffer sbinside = new
StringBuffer();
sbinside. append (inside) ;
getMethodName(sbinside);
return;
}
}
```

```
}
}
if (index3 != -1|| index4 != -1)
{
if (index3 != -1)
Reference =Next. substring (index3+1,
index) ;
else
Reference = Next.substring(index4+1,
index) ;
}
else
Reference = Next.substring(0, index);
if (index2 != -1) //there is a paranthesis
Method = Next.substring(index+1, index2);
else //there is not a paranthesis
Method = Next.substring(index+1);
//get the class name
int counter = ObjectList.size()/2;
String Temp = "";
for (int i 1; i<=counter; i++)
{
Temp =(String) ObjectList.get("Reference" +
i) ;
if (Temp.equalsIgnoreCase(Reference.trim()))
{
Class (String) ObjectList.get("Class"
+ i);
}
```

```
}
if (Class.length()==0) //the reference was not found- it must be static
{
if(Reference.trim() .equalsIgnoreCase("sup
er") )
{
if(CurrentNestedClass.length()>0)
{
Class = NestedExtendedClass;
}
else
{
Class ExtendedClass;
}
}
else if(Reference.trim() .equalsIgnoreCase("this")){
{
if (CurrentNestedClass.length()>0)
{
Class = CurrentNestedClass;
}
else
{
Class CurrentClassName;
}
}
else
{
Class Reference;
```

```
}
}
//insert into db
if ((CurrentNestedClass.length () ! = 0 ||
CurrentClassName . length () ! =0) &&
CurrentMethodName.length() !=0 &&
Class.length() !=0 && Method.length() !=0)
{
DatabaseMethods dataMethods = new
DatabaseMethods();
if (CurrentNestedClass.length()>0)
{
dataMethods.insertCode(CurrentNestedClass
.trim(),
CurrentMethodName.trim(),
Class.trim(), Method.trim());
}
else
{
dataMethods.insertCode(CurrentClassName
.trim(), CurrentMethodName.trim (),
Class.trim(), Method.trim());
}
}
}
else if (Next. contains (" (")) //there is no dot operator
{
String ExtendedClass2 = "";
index = Next.indexOf("(");
```

```
int index1 = -1;
index1 = Next.indexOf("=");
if (index != 0) //has a paranthesis
{
if ((index< index1) && index1 != -1)
Method = Next.substring(0, index);
else if (index1 != -1)
Method = Next.substring(index1+1,
index) ;
else
Method = Next.substring(O, index);
//check for reserve words
if
(Method.trim() .equa1sIgnoreCase("if") ||
Method.trim() .equalsIgnoreCase("catch")
||
Method. trim().equals IgnoreCase (" do ") I||M
ethod.trim() .equalsIgnoreCase("fo
r") ||Method.trim() .equa1sIgnoreCa
se (" return ") || Method. trim () . equal
sIgnoreCase("switch") ||Method.tri
m () . equals IgnoreCase ("while ") )
{
return;
}
if (CurrentNestedC1ass.length()>0)
{
Class = CurrentNestedClass;
ExtendedClass2 =NestedExtendedClass;
```

```
}
else
{
Class = CurrentClassName;
ExtendedClass2 = ExtendedClass;
}
//i want to store all called methods to a
hashmap
int counter = CalledMethods.size() /4;
counter++;
CalledMethods.put("Class" + counter, Class.trim());
CalledMethods.put("Method" + counter, Method.trim());
CalledMethods.put("CurrentMethod" + counter, CurrentMethodName.trim());
CalledMethods.put("ExtendedClass" + counter, ExtendedClass2.trim());
}}}}
```

APPENDIX D

SOURCE CODE : GenerateDiagrams.java

```
import java.io.*;
import java.aet.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
public class GenerateDiagrams extends JPanel
{
public void generateDiagramByClass()
{
DatabaseMethods databaseMethods= new DatabaseMethods();
try
{
//call the database to get the classes and methods
HashMap Results = databaseMethods.getDiagramlnfoByClass();
int size = Results.size()/4;
String CurrentClass = "";
String CurrentMethod = "";
String CalledClass = "";
String CalledMethod = "";
String PreviousClass = "";
String PreviousMethod = "";
String NextCurrentClass = "";
String NextCurrentMethod = "";
String NextCalledClass = "";
String NextCalledMethod = "";
Constants myConstants = new Constants();
```

```
JFrame frame2 = new JFrame("Diagram By Class");
JPanel pClass = new JPanel();
pClass.setLayout(new BoxLayout(pClass, BoxLayout.Y_AXIS));
JPanel pTitle = new JPanel();
JLabel lTitle = new JLabel(new
ImageIcon(myConstants.classTitle));
lTitle.setBorder(BorderFactory.createLineBorder(Color.black));
pTitle.add(lTitle);
pTitle.setBorder(BorderFactory.createEmptyBorder(8,8, 8, 8));
//set up the frame
frame2.setSize(800, 400);
frame2.getContentPane() .setLayout(new BoxLayout (frame2.getContentPane () ,
BoxLayout.Y_AXIS));
frame2.getContentPane().add(pTitle);
for (int i = 1; i<=size; i++)
{
CurrentClass = (String)
Results.get("CurrentClass" + i);
JPanel pCurrentClass = new JPanel();
if (!PreviousClass.equals(CurrentClass))
{
pCurrentClass.setLayout(new BoxLayout(pCurrentClass, BoxLayout.Y_AXIS));
pCurrentClass.setBorder(BorderFactory.createLineBorder(Color.BLACK));
JLabel lCurrentClass = new JLabel(CurrentClass,
SwingConstants.LEFT);
Font labelFont1 = lCurrentClass.getFont();
Font labelFont2 = labelFont1.deriveFont(16.0f);
lCurrentClass.setFont(labelFont2);
pCurrentClass.add(lCurrentClass);
```

```
PreviousMethod = "";
for (int j=i; j<=size; j++)
{
CurrentMethod = (String)Results.get("CurrentMethod"+j ) ;
if(!PreviousMethod.equals(CurrentMethod))
{
JPanel pCurrentMethod  = new JPanel () ;
JLabel lCurrentMethod =new JLabel(CurrentMethod);
Font labelFont3 = lCurrentMethod.getFont();
Font labelFont4 = labelFont3.deriveFont(16. 0f) ;
lCurrentMethod.setFont(labeIFont4);
pCurrentMethod. add (lCurrentMethod);
JLabe1 arrow1 = new
JLabe1(new Imagelcon(myConstants.arrow2));
pCurrentMethod.add(arrow1);
CalledClass = (String)Resu1ts.get("CalledClass" + j);
CalledMethod = (String)Resu1ts.get("CalledMethod" + j);
JPane1 pCalled = new JPane1 () ;
pCalled.setLayout(new BoxLayout(pCalled, BoxLayout.Y_AXIS));
pCalled.setBorder(BorderFactory.createLineBorder(Color. BLACK) ) ;
JLabel lCalled = newJLabel(CalledClass +" ." + CalledMethod);
Font 1abe1Font5 =lCalled.getFont();
Font 1abelFont6 = labelFont5.deriveFont(16.0f);
lCalled.setFont(labelFont6);
pCalled.add(lCalled);
for (int k = j+l; k<=size; k++)
NextCurrentClass=(String) Resu1ts.get("CurrentClass" + k);
NextCurrentMethod=(String) Resu1ts.get("CurrentMethod" +k) ;
NextCalledClass =(String) Resu1ts.get("CalledClass" +k) ;
```

```
NextCalledMethod=(String) Resu1ts.get("CalledMethod" +k) ;

if(NextCurrentClass.equalsIgnoreCase(CurrentClass)&&NextCurrentMethod
.equalsIgnoreCase(CurrentMethod))
{
JLabel lCurrent= new JLabel(NextCalledClass +" " +NextCalledMethod) ;
Font labelFont7= lCalled.get
Font () ;
Font labelFont8= label Font7.deriveFont(16.0f) ;
lCurrent.setFont(labelFont8) ;
pCalled.add(lCurrent) ;
pCurrentMethod.=add (pCalled) ;
j++ ;
}
else
{
break;
}
}
pCurrentMethod.add(pCalled);
pCurrentClass.add(pCurrentMethod) ;
PreviousMethod = CurrentMethod;
PreviousClass = CurrentClass;
if(!NextCurrentClass.equals(CurrentClass))
break;
}
else
{
break;
```

```
}
}
pClass.add(pCurrentClass);
JScrollPane scroll = new JScrollPane(pClass);
scroll.setVerticalScrollBarPolicy(ScrollPaneConstan
ts.VERTICAL_SCROLLBAR_ALWAYS);
frame2.getContentPane() .add(scroll);
//Create and set up the window.
frame2.setDefaultCloseOperation(frame2.DISPOSE_ON_CLOSE) ;
//Display the window.
frame2.pack();
frame2.setVisible(true);
}
catch (Exception e)
{
}
public void generateDiagramByMethod()
{
DatabaseMethods databaseMethods= new DatabaseMethods();
try
{
//call the database to get the classes and methods
HashMap Results =
databaseMethods.getDiagramlnfoByMethod();
int size = Results.size()/4;
String CurrentClass = "";
String CurrentMethod = "";
String CalledClass = "";
String CalledMethod = "";
String PreviousClass = "";
```

```
String PreviousMethod = "";
String NextCurrentClass = "";
String NextCurrentMethod ="";
String NextCalledClass = "";
String NextCalledMethod = "";
Constants myConstants = new Constants();
JFrame frame2 = new JFrame("Diagram By Method");
JPanel pMethod = new JPanel();
pMethod.setLayout(new BoxLayout(pMethod, BoxLayout.Y_AXIS));
JPanel pTitle = new JPanel();
JLabel ITitle = new JLabel(new Imagelcon(myConstants.methodTitle));
ITitle.setBorder (BorderFactory.createLineBorder (Color.black) ) ;
pTitle.add(lTitle);
pTitle.setBorder(BorderFactory.createEmptyBorder(8,8, 8, 8));

//set up the frame
frame2.setSize(800, 400);
frame2.getContentPane() .setLayout(new BoxLayout(frame2.getContentPane(),
BoxLayout.Y_AXIS));
frame2.getContentPane() .add(pTitle);
for (int i = 1; i<=size; i++)
CalledClass = (String) Results.get("CalledClass" + i);
JPanel pCalledClass = new JPanel();
if (!PreviousClass.equals(CalledClass))
{
pCalledClass.setLayout(new
BoxLayout(pCalledClass,BoxLayout.Y_AXIS));
pCalledClass.setBorder(BorderFactory.createLineBorder(Color.BLACK));
JLabel lCalledClass = new JLabel(CalledClass,
```

```
SwingConstants.LEFT);
Font labelFont1 = lCalledClass.getFont();
Font labelFont2 = labelFont1.deriveFont(16.0f);
lCalledClass.setFont(labelFont2);
pCalledClass.add(lCalledClass);
for (int j=i; j<=size; j++)
{
CalledMethod = (String)
Results.get("CalledMethod"+j) ;

if(!PreviousMethod.equals(CalldMethod) )
{
JPanel pCalledMethod= new JPanel();
JLabel lCalledMethod= new JLabel(CalledMethod);
Font labelFont3 = lCalledMethod.getFont();
Font labelFont4= labelFont3.deriveFont(16. 0f) ;
lCalledMethod. setFont (labelFont4);
pCalledMethod. add (lCaledMethod) ;
JLabel arrow 1 = new JLabel(new ImageIcon(myConstants.arrow3));
pCalledMethod.add(arrow1);
CurrentClass = (String)
Results.get("CurrentClass" + j);
CurrentMethod = (String) Results.get("CurrentMethod" + j);
JPanel pCurrent = new JPanel();
pCurrent.setLayout(new BoxLayout(pCurrent, BoxLayout.Y_AXIS));
pCurrent. setBorder (BorderFactory.createLineBorder(Color. BLACK) ) ;
JLabel lCurrent = new JLabel(CurrentClass +"." + CurrentMethod);
Font labelFont5 = lCurrent.getFont();
Font labelFont6 = labelFont5.deriveFont(16.0f) ;
```

```
lCurrent.setFont(labelFont6
} ;
pCurrent.add(lCurrent);
for (int k = j+1; k<=size; k++)
{
NextCurrentClass= (String)Results.get("CurrentClass" + k);
NextCurrentMethod =(String)Results.get("CurrentMethod" + k);
NextCalledClass =(String)Results.get("CalledClass" + k);
NextCalledMethod =(String)Results.get("CalledMethod" +k) ;
if(NextCalledClass.equalsIgnoreCase(CalledClass) &&
NextCalledMethod.equalsIgnoreCase(CalledMethod) )
{
JLabel lNext = new JLabel(NextCurrentClass +"." +NextCurrentMethod);
Font labelFont7=INext.getFont () ;
Font labelFont8=labelFont7.deriveFont (16.0f) ;
INext.setFont(labelFont8);
pCurrent.add(lNext) ;
pCalledMethod.add(pCurrent);
j++;
}
else
break;
}
pCalledMethod. add (pCurrent) ;
pCalledClass.add(pCalledMethod) ;

PreviousMethod = CalledMethod;
if(!NextCalledClass.equals(CalledClass))
break;
```

```
}
}
pMethod.add(pCalledClass) ;
}
PreviousClass = CalledClass;
}
JScrollPane scroll = new JScrollPane(pMethod);
scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
frame2.getContentPane() .add(scroll);
//Create and set up the window.
frame2.setDefaultCloseOperation(frame2.DISPOSE_ON_CLOSE) ;
//Display the window.
frame2.pack();
frame2.setVisible(true);
}
catch (Exception e)
{
}
}}
```

APPENDIX E

SOURCE CODE : MainFrame.java

```
import java.io.*;
import java.awt.*;
import java.awt.event.*
import javax.swing.*;
import javax.swing.filechooser.*;
public class MainFrame extends JPanel implements ActionListener
{
//frame
private JFrame frame=new JFrame("Reverse Engineering Project") ;
//Panels
private JPanel pTitle = new JPanel();
private JPanel pUploaded = new JPanel();
private JPanel pBottom = new JPanel();
private JPanel pBottom2 = new JPanel();
private JPanel pFileChooser = new JPanel();
private JPanel buttonPanel = new JPanel();
private JPanel pGenerator = new JPanel();
//Labels
Constants myConstants = new Constants();
private JLabel lTitle = new JLabel(newImagelcon(myConstants.title));
private JLabel lUploaded = new JLabel("Files Uploaded:",SwingConstants.LEFT);
private JLabel 1generator = new JLabel(newImagelcon(myConstants.generator));

// Buttons
private JButton bReset = new JButton("Reset Application");
private JButton bOpenFile = new JButton("Open File To Read");
//Menu
```

```
private JMenuBar m = new JMenuBar(); // Menubar
private JMenu mFile = new JMenu("File");
private JMenultem miQuit = new JMenultem("Quit");
private JMenu mHelp = new JMenu("Help"); //Help Menu entry
private JMenultem miAbout = new JMenultem("About");
//text areas
private JTextArea tFileList =new JTextArea();
//file chooser
private JFileChooser fc= new JFileChooser();
public MainFrame()
{
//set up the text area
tFileList = new JTextArea(5,20);
tFileList.setMargin(new Insets(5,5,5,5));
tFileList.setEditable(false) ;
JScrollPane logScrollPane = new JScrollPane(tFileList);
//Set menubar
frame.setJMenuBar(m);
//Build Menus
mFile.add(miQuit);
mHelp.add(miAbout);
m.add(mFile);
m.add(mHelp) ;
lTitle.setVerticalAlignment(SwingConstants.TOP);
lTitle.setBorder (BorderFactory.createLineBorder (Color. black) ) ;
lTitle.setForeground(Color.black);
pTitle.add(lTitle);
pTitle.setBorder (BorderFactory.createEmptyBorder (8, 8, 8,8) ) ;
lUploaded.setVerticalAlignment(SwingConstants.CENTER);
```

```
lUploaded.setForeground(Color.black);
Font labelFontl = lUploaded.getFont();
Font labelFont2 = labelFontl.deriveFont(16.0f);
lUploaded.setFont(labelFont2);
pUploaded.add(lUploaded);
pUploaded.setBorder (BorderFactory.createEmptyBorder (8, 8,8, 8));
//set up the the actions
bOpenFile.addActionListener(this);
bGenerateByClass.addActionListener(this);
bGenerateByMethod.addActionListener(this);
bReset.addActionListener(this);
miQuit.addActionListener(new ListenMenuQuit());
//Add Buttons
buttonPanel.add(bOpenFile) ;
pBottom.add(bGenerateByClass);
pBottom.add(bGenerateByMethod);
pBottom2.add(bReset);
pGenerator.add(lgenerator) ;
//set up the frame
frame.setSize(800, 400);

frame.getContentPane() .setLayout(new BoxLayout(frame.getContentPane(),
BoxLayout.Y_AXIS));
frame.getContentPane() .add(pTitle);
frame.getContentPane() .add(buttonPanel);
frame.getContentPane() . add (pFileChooser) ;
frame.getContentPane() .add(pUploaded);
frame.getContentPane() .add(logScrollPane);
frame.getContentPane() . add (pBottom) ;
frame.getContentPane() .add(pBottom2);
```

```
frame.getContentPane() .add(pGenerator);
// Allows the Swing App to be closed
frame.addWindowListener(new ListenCloseWdw());
}
public class ListenMenuQuit implements ActionListener
{
public void actionPerformed(ActionEvent e)
{
System.exit(0);
}
}

public class ListenCloseWdw extends WindowAdapter
{
public void windowClosing(WindowEvent e)
{
System.exit(0);
}
}
public void actionPerformed(ActionEvent e)
{
GenerateDiagrams generate Diagrams = new
GenerateDiagrams();
IIHandle button action.
if (e.getSource() == bOpenFile)
{
int returnVal = fc.showOpenDialog(MainFrame.this);
if (returnVal == JFileChooser.APPROVE OPTION)
{
```

```
File file = fc.getSelectedFile();
tFileList.append(file.getName() + "\n");
FileHandler fileHandler = new FileHandler();
try
{
fileHandler.readFile(file) ;
}
catch (Exception e2)
{
}
}
}
else if (e.getSource() == bGenerateByClass)
{
generateDiagrams.generateDiagramByClass();
}
else if (e.getSource() == bGenerateByMethod)
generateDiagrams.generateDiagramByMethod();
else if (e.getSource() == bReset)
{
DatabaseMethods dataMethods =new
DatabaseMethods();
try
{
dataMethods.resetApplication();
}
catch (Exception e3)
{
}
```

```
private void ShowGUI()
{
//Create and set up the window.
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//Display the window.
frame.pack();
frame.setVisible(true);
}
public static void main(String[] args)
{
MainFrame mf new MainFrame();
mf.ShowGUI();
}
}
```