
A Project Report On Chess cum Chat Application

**Birla Institute of Technology
Mesra (Ranchi)**



**Extension Centre, Jaipur
2004**

Submitted in partial fulfillment for
subject **S**oftware **P**roject **M**anagement

Submitted By:

Aditya Natani
Ajay Kumar Boosar
Vijay Mahawar

Guided By:

Sh. O. P. Rishi

Submitted to:
Birla Institute of Technology, Mesra (Ranchi)
Extension Centre, Jaipur

CERTIFICATE

This is to certify that the following student of BIT, Mesra (Ranchi), Extension Centre, Jaipur of M.C.A. IV Semester have developed a project on "**CHESS CUM CHAT APPLICATION**" (Using Java Applets and Networking).

Students Name

Aditya Natani	(7MCA/5101/02)
Ajay Kumar Boosar	(7MCA/5102/02)
Vijay Mahawar	(7MCA/5128/02)

Sh. O. P. Rishi

(Faculty in-charge SPM and Guide)

(Internal Examiner)

(External Examiner)

ACKNOWLEDGEMENT

The following persons have provided a great deal of moral support and great help without which the project wouldn't have been possible.

1. Sh. O. P. Rishi
2. Sh. B. Pathak
3. Miss. Shweta Mahlawat

We extend our sincere thanks to them for their valuable support, cooperation and time.

We also take this opportunity to thank our friends and batch mates for their valuable suggestions and ideas.

Finally, we would like to convey our heartiest thanks to our family members to adjust with our unconventional and uncommon time schedule; and provide the necessary moral support which was instrumental in the project development.

Developed By
Aditya Natani
Ajay Kumar Boosar
Vijay Mahawar

Signature

CONTENTS

S. No	TOPICS	Page No.
1.	INTRODUCTION	6
1.1.	SCOPE	6
1.2.	OBJECTIVES.....	7
1.3.	SYSTEM ANALYSIS AND DESIGN	8
1.3.1	DEVELOPING A SOLUTION STRATEGY.....	8
1.3.1.1.	FEASIBILITY STUDY	8
1.3.1.2.	ANALYSIS	9
1.3.1.3.	SOLUTION STRATEGY	9
1.3.2.	DESIGN	9
1.3.3.	TESTING	9
1.4.	CHESS GAME: AN INTRODUCTION	11
1.4.1.	OBJECTIVE OF THE GAME.....	12
1.4.2.	THE PIECES AND THE RULES	12
1.4.3.	SOME SPECIAL MOVES AND CASES	16
1.5.	IRC AND WEB CHAT: AN INTRODUCTION	18
2.	OVERVIEW ON PROGRAMMING LANGUAGE USED	20
2.1.	JAVA: AN INTRODUCTION	20
2.1.1.	FEATURES	23
2.1.2.	CLIENT-SERVER ARCHITECTURE.....	31
2.1.3.	DEFINITIONS AND ACRONYMS	40
3.	HARDWARE AND SOFTWARE REQUIREMENTS	47
4.	GUIDE ON USING THE APPLET	48
4.1.	GUIDE TO USE THE CHESS APPLET	48
4.1.1.	GETTING FAMILIAR WITH PIECES	48
4.1.2.	HOW TO MAKE MOVE?	48
4.1.3.	HOW TO CAPTURE OPPONENT PIECE?	48
4.1.4.	HOW TO RECONSIDER A PIECE TO MOVE?	49
4.1.5.	HOW TO PERFORM A CASTLING?	49
4.1.6.	HOW TO PERFORM EN-PASSANT?	49
4.2.	GUIDE TO USE THE CHAT APPLET	49
4.2.1.	HOW TO SEND MESSAGE?	49
5.	DATA FLOW DIAGRAMS	50

6. PROJECT DETAILS	53
6.1. ENHANCEMENTS	53
6.2. PROBLEMS FACED	54
6.3. FUNCTIONING OF VARIOUS PROCESSES	55
6.4. CLASSES AND MEHTODS LIST	43
6.5. SCREENSHOTS	79
6.6. REFERENCES & SOURCES	83
6.6.1. BOOKS	83
6.6.2. WEBSITES	83

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

1. Introduction:

The aim of the project is to provide both a chess as well as chat facility in one application such that two users can play the chess against each other as well as they can chat among themselves on the network that is any number of computers connected via a network. So the users can play the chess in pair and in the meantime when any one of them wants to communicate with the opponent player, he can also do that simultaneously.

1.1. Scope :

Basically this project has two main purposes:

Firstly, to encapsulate the chess gaming and chatting in one application so that a user can easily view and use both of them on a single window.

Secondly, to make these two applications to run simultaneously on the network. Generally, the chess game is normally played against the human user and the computer on which the user is operating, so the need is to make the chess game being played against two human users on two different machines on the network, no matter their physical location and as well as allowing chat among those two users.

1.2. Objectives :

The objectives of this project are:

- a) To provide a user-friendly interactive environment to the users of the application that helps them to play and communicate with a lot of ease.
- b) To provide help to the users in playing the chess that is the different moves of the different pieces etc are being explained to the users, if they require.
- c) The care is taken that the user finds the same chatting mechanism as he is normally used to.
- d) Since there exists client and server as the project is based on client server architecture, where server is serving as a mediator in between the players and the client is making request to server as well as doing all the part that is related to playing logic.
- e) The care has been taken that the application has less CPU usage, so that other applications can also be performed, if required.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jharkhand)

1.3. System Analysis and Design:

1.3.1. Developing a Solution Strategy:

Several solution strategies were outlined with regular constants. Conducting feasibility study for each strategy. Recommending a solution strategy. The steps incorporated in attempting the solution were divided into three phases:

1.3.1.1. Feasibility Study:

Once the program area or solution strategy was identified or set of feasibility study was undertaken to check the validity of available resources. To do the study we need to consider the

- a) Technical Feasibility,
- b) Economical Feasibility and
- c) Behavioural Feasibility.

a) Technical Feasibility:

It is combination of the Software and Hardware requirements. The Hardware requirements comprise of Pc's with minimal requirement and configuration enough to support Graphical User Interface and network establishment (LAN) or Internet available and 2 Button Mouse. The Software requirements are Java Compatible Web Browser and network and Sockets Port.

Requirement Analysis and Specifications:

The different requirements of the project are:

1. A Chess board and its layout & design.
2. The Chess pieces design & moves generation technique and logics.
3. The Chat Applet layout & design.
4. Networking programming logic implementation using Applet designed earlier.
5. Synchronization between the Chess Applet and Chat Applet.

b) Economical Feasibility:

The cost requirement to establish the system should be effective. Cost/Benefit Analysis is made to find out projected Costs and Benefit for successful development of the Application.

SRS Documents:

The different requirements are carefully scrutinized, the moves planned, layout framed and every changes move or message is communicated in real time.

1.3.1.2. Analysis:

It is the detailed study of the various operations or moves of the system. The key question is what must be done to solve the problem.

1.3.1.3. Solution Strategy:

The solution strategy is to develop a layout that comprises of both the chess gaming board and chat window in one frame layout, so that the users can view and use both the application simultaneously. The layout that found most useful was the Java Applet. The other solution strategy was to make the application run over different network and that is done using Java Applets running over the networks using sockets (Socket and Port Programming). The Chess-cum-Chat Application to be used simultaneously is made by embedding Chat Applet with the Chess Applet and using network programming.

1.3.2. Design:

The most creative and challenging phase of the system life cycle is system design. The design is a solution to develop that know-how of the new system under consideration.

It provides the understanding and procedural details necessary for implementing the system recommended in feasibility study. Design phase identifies the boundary and interface of the systems, format I/O etc.

1.3.3. Testing:

Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct the results will be successfully achieved. Effective testing early in the process translates directly into long term cost saving, reduced number of errors. System testing is done when all the modules of the system are in working order and has been tested independently for proper working. All the pieces are put into one system and test to determine, whether it needs user's requirements. The best program is worthless if doesn't needs. System testing is designed to uncover weakness that were not found in earlier tests like program testing in which only syntactical and logical are removed. The purpose of System Testing is to consider all the likely variations to which it will be subjected and then push the system to its limits.

What do you test for?

- 1) The first test for a system is to see whether it produces the correct outputs.
- 2) A volume test is done in which we create as many as records as would normally be produced to verify that the hardware and software will function properly.
- 3) Stress Testing is done to prove that the system doesn't malfunction under peak loads. We subject the system to a high volume of data over a short time period.
- 4) A forced system failure is included to test the backup recovery procedure. Unaccurate data are entered to see the response of the system in terms of error detections and protections and to test that the data and programs are secured from unauthorised access.
- 5) The usability test verifies the user-friendly nature of the system. This related to natural operating and error handling procedures.
- 6) Boundary value analysis is done to find out that the system doesn't produce incorrect result, when the values provided are not within the prescribed domain or at the boundaries of the respective domains.

Types of Testing:

1) Program Testing:

It checks types of errors that is Syntax and Logic.

2) String Testing:

It is used to test a set of related programs forming a subsystem.

3) System Testing:

It test the system as a whole that is, it checks the inter-relationship among the various subsystems.

4) Alpha Testing:

It tests the system in simulated environment on simulated data.

5) Beta Testing:

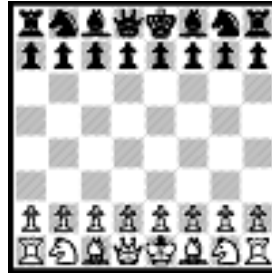
It test the system under realistic conditions with actual data.

6) User Acceptance Testing:

An acceptance testing has the objective of telling the user about the validity and reliability of the system that is the developed system is functioning according to the needs prescribed by the user. This testing is done with the actual data in presence of the user at the users place.

1.4. Chess Game: An Introduction

Chess is a game for two players using a chequered board of sixty-four squares with eight pieces, viz. a king, queen, two rooks, knights, and bishops, and eight pawns each, with the object of placing the opponent's king in checkmate.



The game begins with the position shown above. The White player (the player of the light colored pieces) moves first. Then each player takes a single turn. In fact, a player must move in turn. In other words a move cannot be skipped.

When setting up the pieces, keep in mind two things. The light colored square goes on the player's right, and Queens go on their color next to the Kings on the center files.

You may not move a piece to a square already occupied by one of your own pieces. You may capture an opposing piece by replacing that piece with one of your own pieces, if it can legally move there.


1.4.1. Objective of the Game:

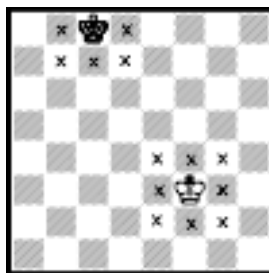
The primary objective in chess is to checkmate your opponent's King. When a King cannot avoid capture then it is checkmated and the game is immediately over. If a King is threatened with capture, but has a means to escape, then it is said to be in check. A King cannot move into check, and if in check must move out of check immediately. There are three ways you may move out of check:

- 1) Capture the checking piece;
- 2) Block the line of attack by placing one of your own pieces between the checking piece and the King. (Of course, a Knight cannot be blocked.);
- 3) Move the King away from check.

If a King is not in check, and no other legal move is possible, then the position is said to be in stalemate. A stalemated game is a draw, or a tie.

1.4.2. The Pieces and The Rules:

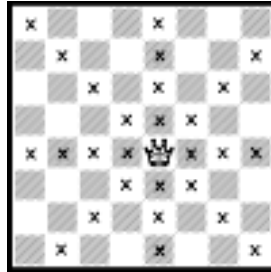
	<p>The KING:</p> <p>The King is the most important piece. When it is trapped so it cannot move without being captured, then the game is lost. This trap is called checkmate.</p>
---	---



The King can move one square in any direction. A King can never move into check, or onto a square where it can be captured by an opponent's piece. If a King is not in check, and no other legal move is possible, then the position is said to be in stalemate. A stalemated game is a draw, or a tie.

**The QUEEN:**

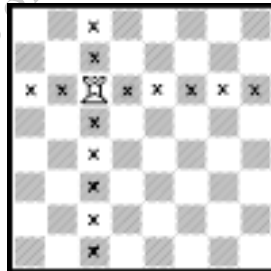
The Queen is the most powerful piece.



The Queen can move to any square in any direction as long as her path is not blocked. Her range and the ability to attack many pieces at once are the source of her power.

**The ROOK:**

The Rook is the next very powerful piece after Queen.

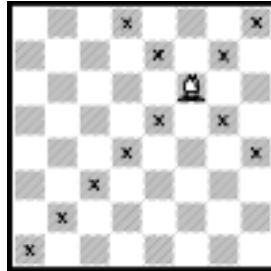


The Rook can move to any square along its file or row as long as its path is not blocked. Its range is the source of its power.



The BISHOP:

The Bishop comes next to Rook in terms of power.

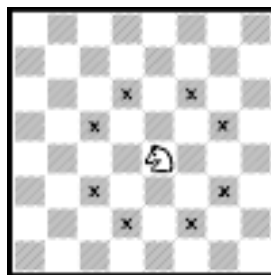


The Bishop can move to any square along its diagonals as long as its path is not blocked. Its range is the source of its power.



The KNIGHT:

The Knight is nearly as powerful as the Bishop.



The Knight is the only piece that can hop over other pieces in an L-shaped path. This ability makes it particularly powerful in the early stage of a game when the board is crowded with pieces.

**The PAWN:**

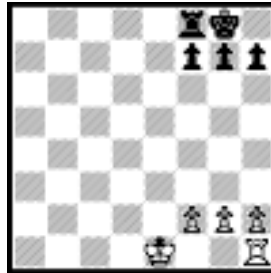
The Pawn is the least powerful piece because of its poor mobility.



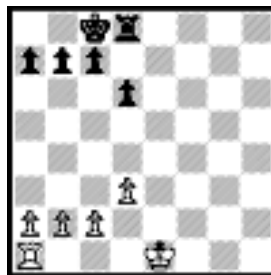
The Pawn may move only one square forward if its path is not blocked. However, it may move as an option one or two squares forward on its first move only. It may capture only diagonally one square. It may not capture forward. It may not move backward. The lowly Pawn usually does not last long, but if it is able to reach the 8th row or rank, then it can promote itself to any other piece except the King. A Pawn thus promoted is replaced by that piece. Therefore, it is possible to have more than one Queen, or two Rooks, Bishops, or Knights on the board at one time.

1.4.3. Some Special Moves and Cases:

1) Castling:



Here Black is castled short or on the King side. White is uncastled.

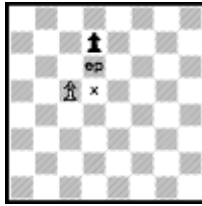


Here Black is castled long or on the Queen side. White is uncastled.

Castling is an important move in chess. It allows a player to quickly move both the King to safety and the Rook to the center for battle. For this reason, wise players carefully guard their ability to castle and usually castle early in the game. Likewise, clever players will attempt to prevent their opponent from castling.

When castling the player moves his King two squares toward one of the player's Rooks and moves that Rook to the opposite side of the King. A player may not castle if either the King or the Rook involved have already moved. Also, the King may not castle out of, through, or into check. There must be no pieces between the King and Rook when castling.

2) En-passant:



A player may capture another player's pawn in passing (En Passant) under very specific circumstances. This move is designed to prevent a player from taking advantage of the two-square first move rule for pawns which might allow them to pass their opponent's pawn(s) without a chance to capture.

The capture is made exactly as if the pawn moved only one square on the first move. In the picture, Black's pawn moved up two squares as is its right. White captured the pawn by removing it from the board and placing the passed white pawn on the square marked ep before playing another move. This move, like any other, is optional and can occur as often as a similar situation arises between pawns.

Submitted to Birla Institute of Technology, Meerut, Kanpur Campus

1.5. IRC and Web Chat: An Introduction

IRC:

IRC is a multi-user chat system that allows many people to communicate simultaneously over the Internet, in real time. It was developed by Jarkko Oikarinen (Finland) in 1988. IRC conversations take place on channels and you chat with other people by typing in messages using the keyboard.

Channels are the virtual locations on IRC networks where users meet to 'talk' to one another. The larger networks have thousands of channels and you have to join one of them before you can talk with other people. It is even possible to have conversations on several channels at the same time. Some channels will be topic specific, but others are less rigid and you will have general chat. Channels have different modes. Most channels are public, but you can talk on a private or secret channel where it is possible to restrict access; for example, to 'invitee only'.

Every channel is run by an operator. You become the operator if you are the first person to join an existing channel that has become empty, or if you create a new channel. You can be made an operator by a person who already has 'op' status. Channel operators have special powers in IRC; they can set the channel mode and control who is allowed on it.

To take part in an IRC you need to run a client program on your computer while connected to the Internet. VSNL users can use the IRC client provided in the Shell account menu. IRC is organized in networks. Each network consists of a series of servers that constantly relay chat back and forth among themselves. You access an IRC by connecting to a specific server and this automatically gives you access to the entire network.

As mentioned above, you need a client program to run an IRC session. If your ISP has not provided you with an IRC client program, you can download any one of the client programs that are available on the web.

Web Chat:

There are many live chat sites on the web. There are hundreds of sites devoted exclusively to chat and many sites offer chat areas as additional feature. Some sites provide links to hundreds of chat servers.

A seasoned IRC user may find the web-based chat rather slow and cumbersome compared to a session run from an IRC client. Many web-chat sites do not show new messages automatically and often you need to scroll through several screens to pick up the thread of a conversation, making it a very difficult task.

On the other hand, a web-based chat is usually more colourful than the IRC. On some sites you can include images and sounds with your messages.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jammu Campus)

2.OVERVIEW ON PROGRAMMING LANGUAGE USED

After a thorough analysis of the problem, several platform alternatives were considered:

- ☒ Visual Basic,
- ☒ VC++,
- ☒ C languages and
- ☒ Java

Java was chosen as appropriate choice due to its platform independence and security measures. Since the application requires a network system therefore Java was an obvious choice.

2.1. Java: An Introduction

Java is a high-level, third-generation programming language, like C, Fortran, Smalltalk, Perl, and many others. It is a platform for distributed computing – a development and run-time environment that contains built-in support for the World Wide Web.

History of Java:

Java development began at Sun Microsystems in 1991, the same year the World Wide Web was conceived. Java's creator, James Gosling, did not design Java for the Internet. His objective was create a common development environment for consumer electronic devices which was easily portable from one device to another. The development team began by using C++, a high-level programming language, as their model but later altered it so as to have simplified syntax, increased robustness, better security features and greater portability across different operating system platforms. This effort evolved into a language, code named Oak and later named Java that retains much of the syntax and power of C++, but is simpler and more platform independent.

Java can be used to write computer applications that crunch numbers, process words, play games, store data or do any of the thousands of other things computer software can do. It is worth while to remember that though java is quite similar to C++. It is a different language with different characteristics.

1. Both programs are composed of one or more files with the "CLASS" extension and having machine independent Java Bytecode.
2. Both require JVM (Java Virtual Machine) to execute these Bytecodes.

The differences between them are as follows:

Applets		Applications
1.	Applets can be embedded in HTML pages and downloaded over the Internet or Intranet.	Applications have no special support in HTML for embedding or downloading.
2.	Applets can only be executed inside a Java compatible container, such as a modern Web Browser.	Applications can be executed from the command line with a small booting utility such as javac.exe or java.exe
3.	Applets execute under strict security limitations that disallow certain operations such as accessing files or systems services on the user's computer.	Applications have no inherent security restrictions.
4.	Applets are the programs written specially for distribution over a network. These programs contain information to be delivered to the world and involve user interaction for example order entry for, registration form, mailing etc.	Applications are system level programs i.e. these programs run in the background and don't involve user interaction, for example server administration, security manager etc.

Applets can be downloaded from the Internet and run safely within a web browser. Traditional computer programs have far too much access to a system and can be downloaded and executed at will. Although you generally trust the maintainers of various FTP archives and bulletin boards to do basic virus checking and not to post destructive software, a lot still slips in through the cracks. (For more details please refer definition and acronyms section)

Java Platform

The evolution of the Java platform and supporting technologies has proceeded at quite a rapid pace. An indication to this quick maturation is that commercial applications have already been developed and are being deployed in Java. Some issues affecting this move to Java are:

- **Reduce Development Time and Cost**

Java takes advantage of OOPS features. Furthermore, it has an edge over C++ for developing robust higher level software quickly.

- **Commercial class libraries**

The Java language includes numerous built-in classes, which forms the foundation for all Java programs.

- **Component Based Applications**

Java is considered to be the foundation to a new breed of productivity applications – such as word processors or spreadsheets – that are based on the object or component-programming model. In this model, applications can be designed as a collections of downloadable components rather than as a single shrink-wrapped software program. This in turn helps to face challenges in performance, usage monitoring and licensing.

The problem with distributing executable programs through web pages is that computer programs are very closely tied to specific hardware and operating system they run on. A Windows programs will not run on a computer that only runs DOS. A Mac application can't run on a Unix workstation. VMS code can't be executed on an IBM mainframe, and so on. Therefore major commercial applications like Microsoft Word and Netscape Navigator have to be written almost separately for all the different platforms they run on. (A platform is loosely defined computer industry buzzword that typically means some combination of hardware and system software that will mostly run all the same software.) Netscape Navigator is one of the most cross-platform compatible of all major applications, yet it only runs on a small number of platforms. Java solves the problem of platform-independence by using byte code. The Java compiler does not produce native executable code for a particular machine like a C compiler would. Instead it produces a special format called byte code. Java byte code written in hexadecimal, byte by byte, looks like this:

```
CA FE BA BE 00 03 00 2D 00 3E 08 00 3B 08 00 01 08 00
20 08
```

This looks a lot like machine language, but unlike machine language, Java byte code is exactly the same on every platform. This byte code fragment means the same thing on a Solaris workstation and a Macintosh PowerBook. Java programs that have been compiled into bytecode, still need an interpreter to execute them on any given platform. The interpreter reads the byte code and translates it into the native language of the host machine on the fly. The most common example of such an interpreter is Sun's program `java` (with a small `j`). Since the bytecode is completely platform independent, only the interpreter and a few native libraries need to be ported to get Java to run on a new computer or operating system. The rest of the runtime environment, including the compiler and most of the class libraries, is written in Java.

All these pieces – the `Javac` compiler, the Java interpreter, the Java programming language, and more are collectively referred to as Java.

2.1.1. Features of the Java Language:

Having seen that Java is equally suited as a language for development both on and off the Internet, it's time to look more closely at the Java language itself. The creators of Java at Sun Microsystems have defined the Java language as "a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, and dynamic language." Well, they managed to fit all of the important 1990s buzzwords into one sentence, but we need to look more closely at Java to see if they managed to fit all of these concepts into one language.

1) SIMPLE

If you have experience with any object-oriented language, especially C++, you probably will find Java to be easier than your high school prom date. Because Java started out as C++ but has had certain features removed, it is certainly a simpler language than C++.

The simplicity of Java is enhanced by its similarities to C and C++. Because many of today's current programmers, especially those likely to consider using Java, are

experienced in at least C and probably C++, Java is instantly familiar to these programmers.

Java has simplified C++ programming by both adding features beyond those found in C++ and by removing some of the features that make C++ a complicated and difficult language to master. Java is simple because it consists of only three primitive data types-numbers, Boolean types, and arrays. Everything else in Java is a class. For example, strings are true objects, not just arrays of characters. Similarly, arrays in the Java language are first-class objects, not just memory allocations and runtime representations.

Java offers additional simplifications over C++. The ubiquitous `goto` statement has been removed. Operator overloading, a frequent source of confusion during the maintenance of C++ programs, is not allowed in Java. Unlike C and C++, Java has no preprocessor. This means that the concepts behind `#define` and `typedef` are not necessary in Java. Java reduces the redundancy of C++ by removing structures and unions from the language. These are both just poor cousins of a full-fledged class and are superfluous in a cohesively designed language. Of course, they were necessary in C++ because it was important for early C++ translators and then compilers to be able to correctly interpret the existing C code that relied on these features.

The most important C++ feature left out of Java is the capability to directly manipulate memory addresses through the use of pointers. Pointers are one of the cornerstones of the C and C++ languages, and it would be difficult to write many programs in these languages without using pointers. However, as any C or C++ programmer will admit, pointers are also a significant source of problems and debugging time in C and C++ programs. Pointers can accidentally be set to point to the wrong thing, causing unexpected behavior including crashes. Pointers also can be used to store allocated memory. If the allocated memory isn't freed, or released back to the operating system, then the program will gradually leak memory, until it eventually runs out. An entire set of commercial products, such as the Bounds Checker products, has come into existence to help programmers identify these types of pointer-related problems. Java simplifies this by completely removing pointers from the language and using a handle-based solution instead.

Of course, if all Java did was remove syntax from C++, it would be a poor compiler instead of an exciting new language. Java goes well beyond C++ by adding some important features. One of the most important is automatic memory management, usually known as *garbage collection*. Garbage collection is really just a blue-collar term that means that you don't need to free memory that you allocate-the Java Virtual Machine takes care of doing this for you. If you're a C or C++ programmer, or have ever had to track down memory leaks in another language, just imagine how nice your life could be if you never have to do it again. You would have time for walks on the beach, barbecued turkey burgers on holiday weekends, and romantic evenings with your spouse.

Java goes beyond C++ in a variety of other ways, as well. For example, Java includes language-level support for writing *multithreaded* programs. A multithreaded program is one that is written such that it performs more than one task at a time. For example, consider the stock price Web page shown earlier in Figure 1.2. One thread in the program to create this page may be constantly retrieving quotes from the stock exchange while another thread searches various news databases for breaking stories about the stocks being monitored. Although you can definitely write this program in a traditional single-threaded manner, the ability to use multiple threads can make it simpler to write and maintain.

2) OBJECT-ORIENTED

Of course, Java is object-oriented. In fact, in the mid-1990s, it's hard to imagine someone developing a new language and declaring it the greatest new thing without it being object-oriented. In its approach to object-orientation, Java follows more closely along the lines of languages such as SmallTalk than C++. Except for its primitive data types, everything in Java is an object. In contrast, C++ is much more lax in that you are entirely free to mix and match object-oriented code (classes) and procedural code (functions). In Java, this is not the case. There are no global functions in Java: all functions are invoked through an object.

Java's support for object-orientation does not include multiple inheritance. The designers of the language felt that the complexity introduced by multiple inheritance was not justified by its benefits.

Java classes are comprised of methods and variables. *Class methods* are the functions that an object of the class can respond to. *Class variables* are the data that define the state of an object. In Java, methods and variables can be declared as *private*, *protected*, or *public*. Private methods and variables are not accessible outside of the class. Protected members are accessible to subclasses of the class, but not to other classes. Finally, public methods and variables are accessible to any class.

Classes in Java can be defined as abstract. An abstract class is a class that collects generic state and behavioral information. More specific classes are defined as subclasses of the abstract class and are used to define actual, specific entities. For example, software in use at a pet store may have an abstract class named Pet. This class would store information that is common to all pets-birthdate, cost, sale price, date received, and so on. Derived from the abstract Pet class could be classes such as Dog, Cat, Bird, and Fish. Each of these classes can augment the abstract class as necessary. For example, a member variable called WaterType (salt or fresh) would be necessary in Fish. Because WaterType would be meaningless for Dogs, Cats, and Birds, it is not part of the abstract implementation of Pet.

3) DISTRIBUTED

Java facilitates the building of distributed applications by a collection of classes for use in networked applications. By using Java's URL (Uniform Resource Locator) class, an application can easily access a remote server. Classes also are provided for establishing socket-level connections.

4) INTERPRETED

Because Java is interpreted, once the Java interpreter has been ported to a specific machine, that machine can instantly run the growing body of Java applications. As an example of the usefulness of this, imagine a hypothetical chip manufacturer, Outtel, that has just finished its newest CPU chip. This new chip, named the Zentium, serves as the foundation of a new line of computers being marketed toward Zen Buddhist monasteries. Once Outtel ports the Java interpreter to work on the Zentium, the new machine will be able to run all of the Java development utilities-the compiler, the debugger, and so on. Contrast this with a traditional language. If Outtel wants to release a C++

compiler with its new computer it must port, or create from scratch, the compiler, the debugger, the runtime library, and so on.

Also, when using an interpreter, programmers are freed from some of the concerns of intermodule dependencies. You no longer have to maintain a "make" file that is sometimes as complicated as the hardest part of your program.

Another advantage is that the time-consuming edit-compile-link-test cycle is broken. Without the compile and link steps, working in an interpreted environment is a much simpler edit-test cycle. Even with today's quick C++ compilers, it is not uncommon for a complete recompile and relink of a large program to be measured in hours and take the better part of a day. Without having to wait for lengthy compiles and links, Java promotes prototyping and easier debugging.

5) ROBUST

The designers of Java anticipated that it would be used to solve some very complex programming problems. Writing a distributed, multithreaded program that can run on a variety of operating systems with a variety of processors is not a simple task. To do it successfully, you need all the help your programming language can offer you. With this in mind, Java was created as a strongly typed language. Data type issues and problems are resolved at compile-time, and implicit casts of a variable from one type to another are not allowed.

Memory management has been simplified in Java in two ways. First, Java does not support direct pointer manipulation or arithmetic. This makes it impossible for a Java program to overwrite memory or corrupt data. Second, Java uses runtime garbage collection instead of explicit freeing of memory. In languages like C++, it is necessary to delete or free memory once the program has finished with it. Java follows the lead of languages such as LISP and SmallTalk by providing automatic support for freeing memory that has been allocated but is no longer used.

6) SECURE

Closely related to Java's robustness is its focus on security. Because Java does not use pointers to directly reference memory locations, as is prevalent in C and C++, Java has a great deal of control over the code that exists within the Java environment.

It was anticipated that Java applications would run on the Internet and that they could dynamically incorporate or execute code found at remote locations on the Internet. Because of this, the developers of Java hypothesized the existence of a hostile Java compiler that would generate Java byte codes with the intent of bypassing Java's runtime security. This led to the concept of a byte-code verifier. The byte-code verifier examines all incoming code to ensure that the code plays by the rules and is safe to execute. In addition to other properties, the byte code verifier ensures the following:

No pointers are forged.

No illegal object casts are performed.

There will be no operand stack overflows or underflows.

All parameters passed to functions are of the proper types.

Rules regarding private, protected, and public class membership are followed.

7) ARCHITECTURE-NEUTRAL

Back in the dark ages of the early 1980s, there was tremendous variety in desktop personal computers. You could buy computers from Apple, Commodore, Radio Shack, Atari, and eventually even from IBM. Additionally, every machine came with its own very different operating system. Because developing software is such a time-consuming task, very little of the software developed for use on one machine was ever ported and then released for use on a different machine.

In many regards, this situation has improved with the acceptance of Windows, the Apple Macintosh, and UNIX variations as the only valid personal computer options. However, it is still not easy to write an application that can be used on Windows NT, UNIX, and a Macintosh. And it's getting more complicated with the move of Windows NT to non-Intel CPU architectures.

A number of commercially available source code libraries (for example, Zinc, ZApp, and XVT) attempt to achieve application portability. These libraries attempt this by focusing on either a lowest common denominator among the operating systems or by creating a common core API (Application Programming Interface).

Java takes a different approach. Because the Java compiler creates byte code instructions that are subsequently interpreted by the Java interpreter, architecture neutrality is achieved in the implementation of the Java interpreter for each new architecture.

8) PORTABLE

In addition to being architecture-neutral, Java code is also portable. It was an important design goal of Java that it be portable so that as new architectures (due to hardware, operating system, or both) are developed, the Java environment could be ported to them.

In Java, all primitive types (integers, longs, floats, doubles, and so on) are of defined sizes, regardless of the machine or operating system on which the program is run. This is in direct contrast to languages like C and C++ that leave the sizes of primitive types up to the compiler and developer. Additionally, Java is portable because the compiler itself is written in Java and the runtime environment is written in POSIX-compliant C.

9) HIGH-PERFORMANCE

For all but the simplest or most infrequently used applications, performance is always a consideration. It is no surprise, then, to discover that achieving high performance was one of the initial design goals of the Java developers. A Java application will not achieve the performance of a fully compiled language such as C or C++. However, for most applications, including graphics-intensive ones such as are commonly found on the World Wide Web, the performance of Java is more than adequate. For some applications, there may be no discernible difference in performance between C++ and Java.

Many of the early adopters of C++ were concerned about the possibility of performance degradation as they converted their programs from C to C++. However, many C++ early adopters discovered that, although a C program will outperform a C++ program in many cases, the additional development time and effort don't justify the minimal performance gains. Of course, because we're not all programming in assembly language, there must be some amount of performance we're willing to trade for faster development.

It is very likely that early experiences with Java will follow these same lines. Although a Java application may not be able to keep up with a C++ application, it will normally be fast enough, and Java may enable you to do things you couldn't do with C++.

10) MULTITHREADED

Writing a computer program that only does a single thing at a time is an artificial constraint that we've lived with in

most programming languages. With Java, we no longer have to live with this limitation. Support for multiple, synchronized threads is built directly into the Java language and runtime environment.

Synchronized threads are extremely useful in creating distributed, network-aware applications. Such an application may be communicating with a remote server in one thread while interacting with a user in a different thread.

11) DYNAMIC

Because it is interpreted, Java is an extremely dynamic language. At runtime, the Java environment can extend itself by linking in classes that may be located on remote servers on a network (for example, the Internet). This is a tremendous advantage over a language like C++ that links classes in prior to runtime.

In C++, every time member variables or functions are added to a class, it is necessary to recompile that class and then all additional code that references that class. Of course, the problem is exacerbated by the fact that you need to remember to recompile the files that reference the changed class. Using make files reduces the problem, but for large, complex systems, it doesn't eliminate it.

Java addresses this problem by deferring it to runtime. At runtime, the Java interpreter performs name resolution while linking in the necessary classes. The Java interpreter is also responsible for determining the placement of objects in memory. These two features of the Java interpreter solve the problem of changing the definition of a class used by other classes. Because name lookup and resolution are performed only the first time a name is encountered, only minimal performance overhead is added.

2.1.2. Client-Server architecture:

An Introduction to Sockets

The computers on the Internet are connected by the TCP/IP protocol. In the 1980s, the Advanced Research Projects Agency (ARPA) of the U.S. government funded the University of California at Berkeley to provide a UNIX implementation of the TCP/IP protocol suite. What was developed was termed the *socket interface*, although you might hear it called the Berkeley-socket interface or just Berkeley sockets. Today, the socket interface is the most widely used method for accessing a TCP/IP network.

A socket is nothing more than a convenient abstraction. It represents a connection point into a TCP/IP network, much like the electrical sockets in your home provide a connection point for your appliances. When two computers want to converse, they each use a socket. One computer is termed the server-it opens a socket and listens for connections. The other computer is termed the client; it calls the server socket to start the connection. To establish a connection, all that's needed is a destination address and a port number.

Each computer in a TCP/IP network has a unique address. *Ports* represent individual connections within that address. This is analogous to corporate mail-each person within a company shares the same address, but a letter is routed within the company by the person's name. Each port within a computer shares the same address, but data is routed within each computer by the port number. When a socket is created, it must be associated with a specific port-this is known as binding to a port.

Socket Transmission Modes

Sockets have two major modes of operation: *connection-oriented* and *connectionless*. Connection-oriented sockets operate like a telephone; they must establish a connection and a hang up. Everything that flows between these two events arrives in the same order it was sent. Connectionless sockets operate like the mail-delivery is not guaranteed, and multiple pieces of mail may arrive in a different order than they were sent.

Which mode to use is determined by an application's needs. If reliability is important, then connection-oriented operation is better. File servers need to have all their data arrive correctly and in sequence. If some data was lost, the server's usefulness would be invalidated. Some applications-a time server, for example-send discrete chunks of data at regular intervals. If the data became lost, the server would not want the network to retry until the data was sent. By the time the data arrived, it would be too old to have any accuracy. When you need reliability, be aware that it does come with a price. Ensuring data sequence and correctness requires extra processing and memory usage; this extra overhead can slow down the response times of a server.

Connectionless operation uses the User Datagram Protocol (UDP). A datagram is a self-contained unit that has all the information needed to attempt its delivery. Think of it as an envelope-it has a destination and return address on the outside and contains the data to be sent on the inside. A socket in this mode does not need to connect to a destination socket; it simply sends the datagram. The UDP protocol promises only to make a best-effort delivery attempt. Connectionless operation is fast and efficient, but not guaranteed.

Connection-oriented operation uses the Transport Control Protocol (TCP). A socket in this mode needs to connect to the destination before sending data. Once connected, the sockets are accessed using a streams interface: open-read-write-close. Everything sent by one socket is received by the other end of the connection in exactly the same order it was sent. Connection-oriented operation is less efficient than connectionless, but it's guaranteed.

Sun Microsystems has always been a proponent of internetworking, so it isn't surprising to find rich support for sockets in the Java class hierarchy. In fact, the Java classes have significantly reduced the skill needed to create a sockets program. Each transmission mode is implemented in a separate set of Java classes. The connection-oriented classes will be discussed first.

Java Connection-Oriented Classes

The connection-oriented classes within Java have both a client and a server representative. The client half tends to be the simplest to set up, so it will be covered first.

Listing 9.1 shows a simple client application. It requests an HTML document from a server and displays the response to the console.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

A simple socket client.

```
import java.io.*;
import java.net.*;

/**
 * An application that opens a connection to a Web server and reads
 * a single Web page from the connection.
 * NOTE: "merlin" is the name of my local machine.
 */
public class SimpleWebClient {
    public static void main(String args[])
    {
        try
        {
            // Open a client socket connection
            Socket clientSocket1 = new Socket("merlin", 80);
            System.out.println("Client1: " + clientSocket1);

            // Get a Web page
            getPage(clientSocket1);
        }
        catch (UnknownHostException uhe)
        {
            System.out.println("UnknownHostException: " + uhe);
        }
        catch (IOException ioe)
        {
            System.err.println("IOException: " + ioe);
        }
    }

    /**
     * Request a Web page using the passed client socket.
     * Display the reply and close the client socket.
     */
    public static void getPage(Socket clientSocket)
    {
        try
        {
            // Acquire the input and output streams
            DataOutputStream outbound = new DataOutputStream(
                clientSocket.getOutputStream() );
            DataInputStream inbound = new DataInputStream(
                clientSocket.getInputStream() );

            // Write the HTTP request to the server
            outbound.writeBytes("GET / HTTP/1.0\r\n\r\n");

            // Read the response
            String responseLine;
            while ((responseLine = inbound.readLine()) != null)
            {
                // Display each line to the console
                System.out.println(responseLine);

                // This code checks for EOF. There is a bug in the

```

```
// socket close code under Win 95. readLine() will
// not return null when the client socket is closed
// by the server.
if ( responseLine.indexOf("</HTML>") != -1 )
    break;
}

// Clean up
outbound.close();
inbound.close();
clientSocket.close();
}
catch (IOException ioe)
{
    System.out.println("IOException: " + ioe);
}
}
```

Recall that a client socket issues a connect to a listening server socket. Client sockets are created and connected by using a constructor from the Socket class. The following line creates a client socket and connects it to a host:

```
Socket clientSocket = new Socket("merlin", 80);
```

The first parameter is the name of the host you want to connect to; the second parameter is the port number. A host name specifies only the destination computer. The port number is required to complete the transaction and allow an individual application to receive the call. In this case, 80 was specified, the well-known port number for the HTTP protocol. Other well-known port numbers are shown in Table 9.1. Port numbers are not mandated by any governing body, but are assigned by convention-this is why they are said to be "well known."

List of Well-known port numbers.

<i>Service</i>	<i>Port</i>
echo	7
daytime	13
ftp	21
telnet	23
smtp	25
finger	79
http	80
pop3	110

Because the `Socket` class is connection oriented, it provides a streams interface for reads and writes. Classes from the `java.io` package should be used to access a connected socket:

```
DataOutputStream outbound = new
    DataOutputStream (clientSocket.getOutputStream());
DataInputStream inbound = new
    DataInputStream (clientSocket.getInputStream());
```

Once the streams are created, normal stream operations can be performed:

```
outbound.writeBytes("GET / HTTP/1.0\r\n\r\n");
String responseLine;
while ( (responseLine = inbound.readLine()) != null)
{
    System.out.println(responseLine);
}
```

The above code snippet requests a Web page and echoes the response to the screen. When the program is done using the socket, the connection needs to be closed:

```
outbound.close();
inbound.close();
clientSocket.close();
```

Notice that the socket streams are closed first. All socket streams should be closed before the socket is closed. This application is relatively simple, but all client programs follow the same basic script:

- 1) Create the client socket connection.
- 2) Acquire read and write streams to the socket.
- 3) Use the streams according to the server's protocol.
- 4) Close the streams.
- 5) Close the socket.

Using a server socket is only slightly more complicated, as explained in the following section.

A simple server application.

```
/**
 * An application that listens for connections and serves a simple
 * HTML document.
 */
class SimpleWebServer {
    public static void main(String args[])
    {
        ServerSocket serverSocket = null;
        Socket clientSocket = null;
        int connects = 0;
        try
        {
            // Create the server socket
            serverSocket = new ServerSocket(80, 5);

            while (connects < 5)
            {
                // Wait for a connection
                clientSocket = serverSocket.accept();

                //Service the connection
                ServiceClient(clientSocket);
                connects++;
            }
            serverSocket.close();
        }
        catch (IOException ioe)
        {
            System.out.println("Error in SimpleWebServer: " + ioe);
        }
    }

    public static void ServiceClient(Socket client)
        throws IOException
    {
        DataInputStream inbound = null;
        DataOutputStream outbound = null;
        try
        {
            // Acquire the streams for IO
            inbound = new DataInputStream( client.getInputStream());
            outbound = new DataOutputStream( client.getOutputStream());

            // Format the output (response header and tiny HTML document)
            StringBuffer buffer = PrepareOutput();

            String inputLine;
            while ((inputLine = inbound.readLine()) != null)
            {
                // If end of HTTP request, send the response
                if ( inputLine.equals("") )
                {
                    outbound.writeBytes(buffer.toString());
                    break;
                }
            }
        }
    }
}
```

```
    }  
    finally  
    {  
        // Clean up  
        System.out.println("Cleaning up connection: " + client);  
        outbound.close();  
        inbound.close();  
        client.close();  
        client.close();  
    }  
}
```

Servers do not actively create connections. Instead, they passively listen for a client connect request and then provide their services. Servers are created with a constructor from the `ServerSocket` class. The following line creates a server socket and binds it to port 80:

```
ServerSocket serverSocket = new ServerSocket(80, 5);
```

The first parameter is the port number on which the server should listen. The second parameter is optional. The API documentation indicates that this parameter is a listen time, but in traditional sockets programming the listen function's second parameter is the listen stack depth. As it turns out, this is also true for the second constructor parameter. A server can receive connect requests from many clients at the same time, but each call must be processed one at a time. The *listen stack* is a queue of unanswered connect requests. The above code instructs the socket driver to maintain the last five connect requests. If the constructor omits the listen stack depth, a default value of 50 is used.

Once the socket is created and listening for connections, incoming connections are created and placed on the listen stack. The `accept()` method is called to lift individual connections off the stack:

```
Socket clientSocket = serverSocket.accept();
```

This method returns a connected client socket used to converse with the caller. No conversations are ever conducted over the server socket itself. Instead, the server socket will spawn a new socket in the `accept()` method. The server socket is still open and queuing new connection requests.

Like the client socket, the next step is to create an input and output stream:

```
DataInputStream inbound = new  
    DataInputStream (clientSocket.getInputStream());
```

```
DataOutputStream outbound = new  
    OutputStream( clientSocket.getOutputStream());
```

Normal I/O operations can now be performed by using the newly created streams. This server waits for the client to send a blank line before sending its response. When the conversation is finished, the server closes the streams and the client socket. At this point, the server tries to accept more calls. What happens when there are no calls waiting in the queue? The method will wait for one to arrive. This behavior is known as *blocking*. The `accept()` method will block the server thread from performing any other tasks until a new call arrives. When five connects have been serviced, the server exits by closing its server socket. Any queued calls will be canceled.

All servers follow the same basic script:

- 1) Create the server socket and begin listening.
- 2) Call the `accept()` method to get new connections.
- 3) Create input and output streams for the returned socket.
- 4) Conduct the conversation based on the agreed protocol.
- 5) Close the client streams and socket.
- 6) Go back to step 2, or continue to step 7.
- 7) Close the server socket.

2.1.3. Definitions & Acronyms:

Transmission Control Protocol/Internet Protocol reserve lower 1024 ports for specific protocols and the higher are free and can be used for the other purposes. Like port no. 21 is reserved for FTP. Port no. 23 for telnet Port no. 25 for email Port no. 119 for net users. It is up to protocol to determine how a client should interact with port.

◎ **HTTP:**

It is a protocol that web users and servers use to transfer Hyper Text Pages and images.

◎ **Proxy Server:**

A proxy server speaks the client side of a protocol to another server. A proxy server has a additional ability to filter certain requests or cache the results for specific requests that may come in future.

◎ **Threads:**

Thread are one of the most important aspects of our application. Since our application demand simultaneous processing of several tasks and so the threads are used. Moreover, java provides a very good mechanism of creating threads.

Ways to create a new thread:

Method 1:

Extend the thread class in your class and override the method run.

```
public class mythread extends thread
{
    public void run()
    {
        .....
        .....
    }
}
```


Method 2:

Implementing the runnable interface (the class thread itself is an implementation of runnable) and use that class in thread construction.

```
public class mythread implements Runnable
{
    public void run()
    {
        .....
        .....
    }
}
```

```
Thread t1= new Thread(new mythread);
```

The general thread methods used are:

```
wait();
notify();
notifyall();
```

All these methods can be called only with in a synchronized method.

Options:

```
ThreadGroup(String groupName)
```

```
ThreadGroup(ThreadGroup parentobject,String groupName)
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

© **InetAddress class:**

It is used to encapsulate both the numerical IP address and the domain name for that address. The InetAddress class has no visible constructors. We have to use the factory methods.

Factory Methods:

They are merely a convention where by static method in a class return a instance of that class.

```
static InetAddress getLocalHost()  
static InetAddress getByName(String hostName)  
static InetAddress getByName(String hostName)
```

Note: The above three methods throw UnknownHostException

Instance Methods:

```
boolean equals()  
byte getAddress()  
String getHostAddress()  
String getHostName()  
int hashCode()  
boolean is MulticastAddress()  
String toString()
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jharkhand Campus)

© **TCP/IP Client Sockets:**

There are two kinds of TCP Sockets:

- a. Server Socket (Designed to be a listener, which wait for client before doing anything)
- b. Socket (Designed to connect to the server socket and initiate protocol exchange)

Two constructors are used to create client socket:

- a. Socket (String hostName, int port):
It creates a socket connecting the local host. The port can throw UnknownHostException or an IOException
- b. Socket(InetAddress ipAddress, int port) throws IOException

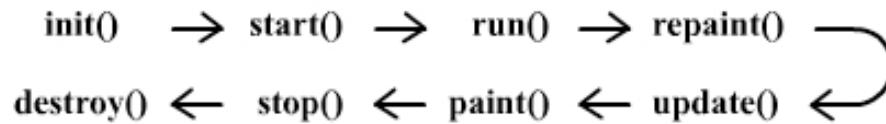
Methods:

- I InetAddress getInetAddress()
Returns the InetAddress associated with the socket object.
- II int getport()
Returns the remote port to which this socket object is connected.
- III int getLocalport()
Returns the local port.

© **Applet:**

An applet is a program that is designed to be dynamically downloaded across the network and run by a java compatible web browser. It is a dynamic program that is smart enough to change according to the inputs and presents different output.

An Applet program has a pre-defined path of execution within which the entire processing has to be done. An applet receives its start from the Applet class so an applet does not implement a main method. The default sequence of an applet's method is:



The Basic Applet Life Cycle

Java solves the problem of security by severely restricting what an applet can do. A Java applet cannot write to your hard disk without your permission. It cannot write to arbitrary addresses in memory and thereby introduce a virus into your computer. It should not crash your system. These are also known as **TRUSTED APPLETS** (Microsoft Java does not support this feature of distinguishing between trusted and untrusted Applets whereas Sun Java does). The specification for JDK 1.1 and onwards includes a new package, The Java Security API, that includes support for digital signature and code signing. Microsoft Internet Explorer can distinguish between trusted and untrusted applets through the use of digital signatures.

Browsers for Your Applets

Browsers are the most popular type of external viewer for applets. Think of browsers as your windows to the Web; change your browser and you get a whole new view of what is out there. Browsers are available for virtually any computer operating system from DOS to Mac to OS/2.

When you set up your intranet, you want to choose a Java-capable browser. The number of Java-capable browsers is steadily increasing. Currently, the most popular browsers that support Java include

- HotJava
- Microsoft Internet Explorer
- Netscape Navigator
- Oracle PowerBrowser

HotJava

Not only is HotJava the first browser to support Java, it is also the first browser written entirely in Java. HotJava is being developed by JavaSoft. To say that JavaSoft is developing HotJava slowly is an understatement. Until April 1996, the alpha version of HotJava was the only version available. Unfortunately, HotJava alpha cannot run applets written in beta or later versions of Java.

The current version of HotJava is a beta version. Fortunately, this version supports JDK 1.0 and later. Versions of HotJava are available for Windows 95/NT, Sun Solaris, and soon Macintosh.

Because HotJava is the first Java-capable browser, it is a popular browser. This popularity leads to disappointment for some new users, especially because HotJava is not feature-rich like some of the other Java-capable browsers. You find that HotJava has a rather plain interface and limited extras. Still, HotJava is currently in the testing stages and may yet evolve into a full-featured browser.

Internet Explorer

The Internet Explorer is a feature-rich browser from Microsoft. As one of the most powerful browsers currently available, Internet Explorer took the Web by storm when it was first released in 1995 and quickly moved to the number two browser on the market. You can obtain free versions of Internet Explorer for Macintosh, Windows 95, and Windows NT.

The popularity of Internet Explorer stems largely from its support for existing HTML standards and unique extensions to HTML. Internet Explorer supports HTML 3.2; all Netscape 1.0 and 2.0 extensions; and powerful multimedia extensions including document soundtracks, scrolling marquees, and inline video. Versions 3.0 and later also feature support for Java and ActiveX. As of July 1996, Internet Explorer was also the only browser to fully support the expanded HTML table model specification.

Netscape Navigator

Although the Internet Explorer is vying with the Netscape Navigator for its coveted position as king of the browsers, the Navigator remains the hands-on favorite. The Netscape Navigator is available for Macintosh, Windows, Windows 95/NT, and UNIX.

Netscape Navigator supports Java and has many features that make it a great choice for your intranet. These features include support for HTML 3.2, plug-ins, and JavaScript. With HTML 3.2, you get support for the advanced features of HTML like tables and client-side image maps. Plug-ins allow you to add modules for inline video, sound, and multimedia. Using JavaScript, you can create client-side scripts for your HTML documents.

Netscape has also introduced unique extensions to HTML with every major release of the Navigator. Currently, Netscape and Microsoft are playing a game of one-upmanship. Netscape Navigator is the first browser to support HTML tables and is the model for the table standard adopted in HTML 3.2. By supporting the expanded HTML table model specification, Internet Explorer went one better in version 3.0. Netscape Navigator 2.0 introduced frames, which are mini-windows within documents. Internet Explorer 3.0 went one better and introduced frames without borders and frames that can float on the page. Netscape added new extensions to the beta release of Navigator 3.0 that include support for frames without borders.

Oracle PowerBrowser

Although PowerBrowser is a fairly new browser on the market, it has all the features you expect in a browser created by Oracle. Oracle is known for its powerful databases and not surprisingly, PowerBrowser includes a local database called Blaze. With Blaze, you can store and manage large amounts of data efficiently. Because Blaze and PowerBrowser can communicate, you can easily create HTML pages that access the Blaze database. Other features of the browser include support for HTML 3.2 and Java.

Currently, PowerBrowser is available for Windows 3.1 and Windows 95/NT.

3. Hardware & Software Requirements:

The application must be capable of reflecting the changes made to the chess board positions of the different pieces, when a user makes a move or when some message is sent via a chat application to other machine. All these changes must be reflected in real time and without any delay that is according to the human reaction time. It also needs to be multi-threaded to cater to the needs of different users at the same time. Hence, the requirements are a bit on the higher side.

- ✓ It ideally requires a processor running at above 2 GHz frequency.
- ✓ It must be having 128 MB of RAM or higher.
- ✓ The hard disk must be 4 GB .
- ✓ The LAN or any other network should be established and LAN card must be there on the machines where application will be used
- ✓ The pointing device must be an optical mouse.
- ✓ The system runs effectively on Windows 2000 server but it will also run equally well on compatible operating systems.
- ✓ The web browser must be Microsoft Internet Explorer with a resolution of at least 800 * 600.

Submitted to Birla Institute of Technology (Vellore Campus)

4. GUIDE ON USING THE APPLET:

4.1. Guide to use the Chess Applet:

The Chess Applet consists of 64 squares. With 32 squares initially occupied by the pieces as discussed in the introductory section. The Chess Applet is fully mouse oriented one. This guide has been divided into various sections for user convenience. They are as follows:

4.1.1. Getting familiar with pieces



4.1.2. How to make move (It is a two-click process):

The chess game is fully mouse oriented and key board is not functional. The user can make a legal move by performing following operations.

First Click: On the square containing the piece which the player wants to move. (The player can move only his piece), the square gets highlighted by RED colour.

Second Click: The square where he want the piece to move.

4.1.3. How to capture opponent piece:

First Click: On the square containing the piece which the player wants to move. (The player can move only his piece), the square gets highlighted by RED colour.

Second Click: The square containing the opponent piece which the player wants to capture.

4.1.4. How to reconsider a piece to move (Deactivating the selected piece):

Once the user has clicked on the piece to move and rethinks on the piece to move, for this he can click on any of his other piece, the highlighted square will be de-activated and he can start over again.

Single Click on any of your own piece.

4.1.5. How to perform a castling:

To make any castling, whether o-o (short castling) or o-o-o (long castling) the user has to perform following operation:

First Click: On the square containing the King.

Second Click: On the square where the king will be placed after castling.

4.1.6. How to perform an en-passant:

To achieve this, user has to perform following operation:

First Click: On the square containing the Pawn.

Second Click: On the square skipped by the Opponent Pawn.

4.2. Guide to Use the Chat Applet:

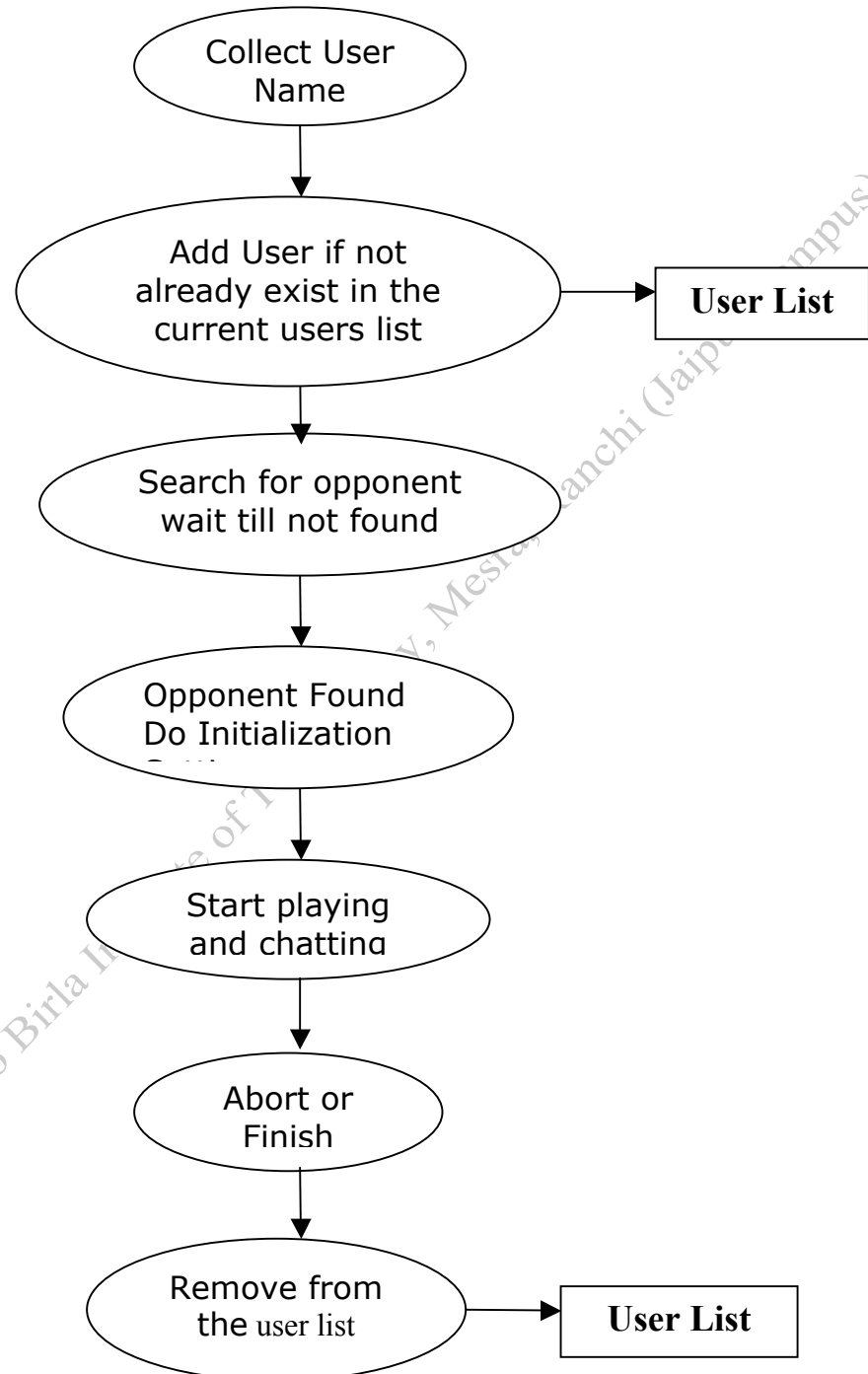
The Chat Applet consists of a textbox and a text area. It is fully keyboard oriented one. The textbox is used to get input from the user/player, it is editable. The text area is used to display the message. It is not editable i.e. user cannot make any changes in it.

4.2.1. How to Send Message?

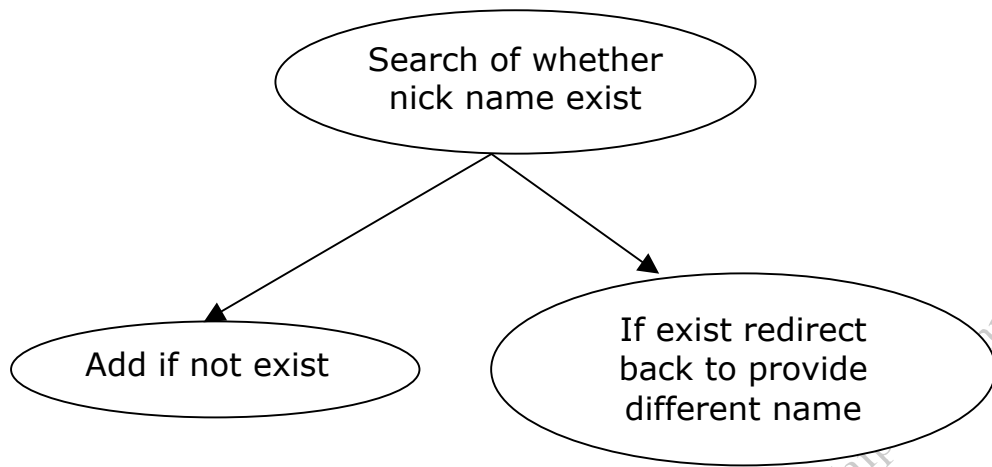
A user has to type in the text in the 'send' textbox and press return key when finished. The message gets displayed in the text area immediately.

5. DATA FLOW DIAGRAMS:

First Level DFD:



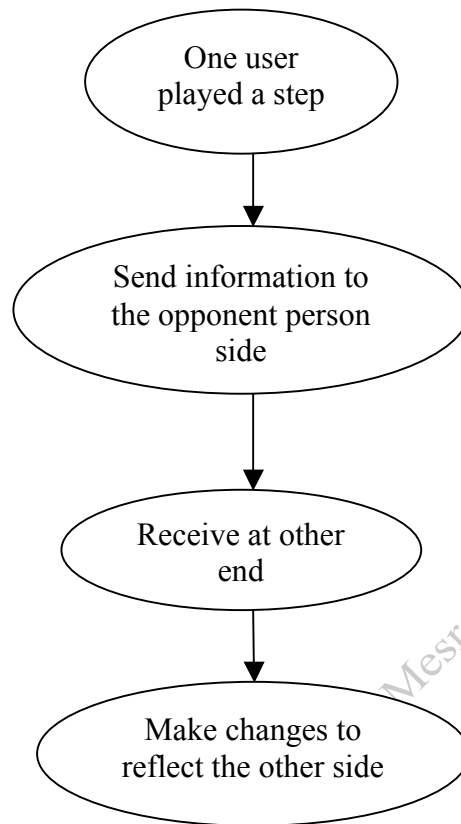
Second Level DFD:



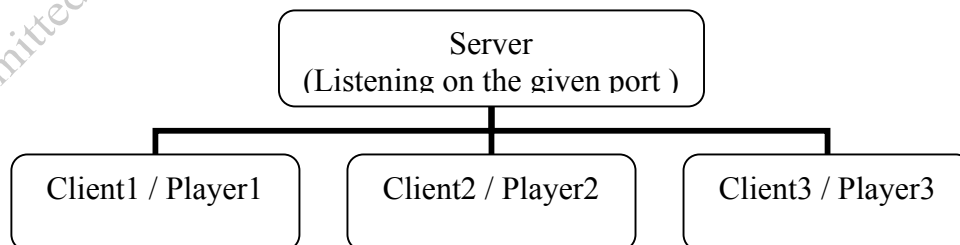
THIRD LEVEL DFD:



FOURTH LEVEL DFD:



Grouping at the Sever End:



6. Project Details:

6.1. Enhancements:

- a) To make the application implemented on mobile.
- b) To make the chess moves to be implemented using mouse dragging.
- c) To add smileys in the chat option.
- d) To make the chess application single user against the machine.
- e) To change the theme of the pieces and board
- f) To enable file transfer in chat application.
- g) To give the option of saving the player moves (i.e. importing and exporting move history using file formats *.pgn and *.fen)
- h) Conducting Tournaments.
- i) Storing and maintaining the player records (win and lost percentage)
- j) To view the ongoing games in process.
- k) To implement the total time taken by each opponent in making the moves in the end.
- l) To add the tutorial section for the naive users.

Submitted to Birla Institute of Technology, Mesra Ranchi (Jaipur Campus)

6.2. Problem Faced:

1. Designing the board: whether to take a picture or drawing the board using Graphics and java.awt classes
2. Move to be updated in opponents PC.
3. Alternatively changing turns of players.
4. A player to move only his pieces.
5. A white player has white at bottom and black having black.
6. Displaying nickname at appropriate place.
7. Pairing of players
8. Maintaining the status of the player at the server.
9. When updating the board after move we got stuck into a serious problem in passing the graphics object to the related functions which was finally solved by invoking all the related functions from update function itself.
10. At the end of the project the updation became a tedious job as the flow of control was hazy and the logic has to be understood right from scratch.
11. How to identify the move is coming from which source, whether it is server, chessboard, chat box or from the opponent special cases.
12. The difficulty was faced while implementing special chess moves like castling, en-passant etc. which was solved by defining separate protocol in the portTalk class of the chessBoard.

6.3. Functioning of various processes:

UNDERLYING PROCESS (Server Side):

The actual process behind the application is that two different threads are running, one will allow the chess play and the other will perform the chatting task.

Chatting in itself has two processes:

- a. Listen
- b. Send

The threading performs the following tasks:

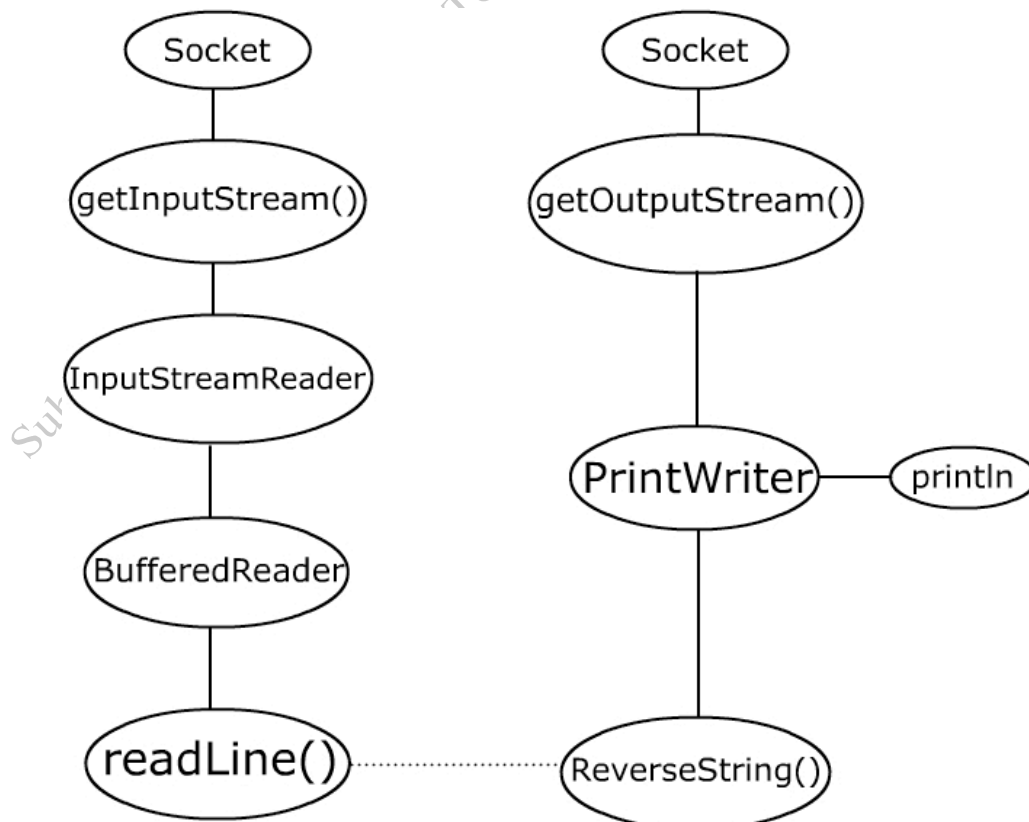
- a. **Chess Moves:** Move details with other information including special cases like Castling, En-Passant, Board etc.
- b. **Message:** Chatting related information.
- c. Special Message supplied by the server for starting, ending and giving some special messages.

Buffered Reader & Print Writer:

These classes are used for reading and writing data on sockets.

Server Socket:

It establishes the client and server communication.



The project makes heavy use of java and its applications (applets, socket programming and threads) and HTML.

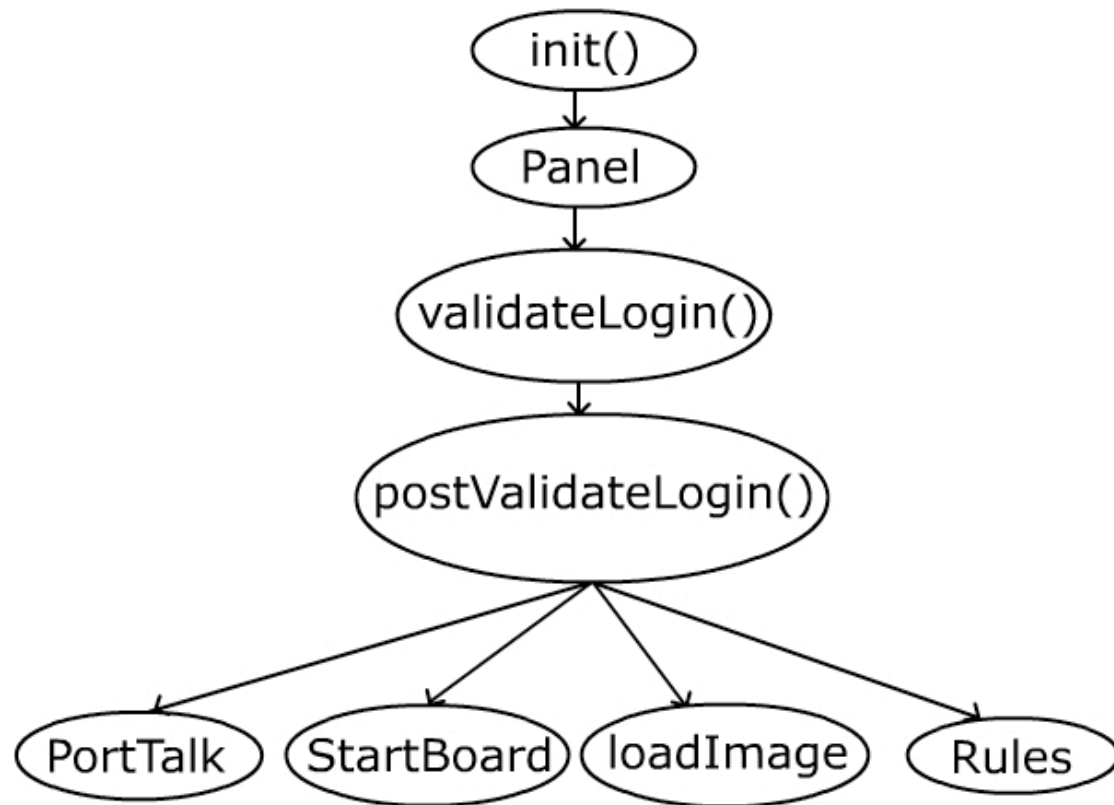
We have divided our application in two parts. Client and Server. Server keeps on waiting for the client request at some port let us say 1234. When client sends a request server accepts the request and start a thread for the client. Then server transfers the client to some other temporary port and again begins the same process i.e. waiting for the other client. Mean while it keeps track of all the clients and make the pairs so that two player / client can start playing chess.

On the other end when one player gets information about the opponent the game begins. Along with the game we have also provided the facility of the chatting. Chessboard heavily utilizes the graphics of the java.awt, java.awt.event and applet functionality of the java.applet package.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jharkhand Campus)

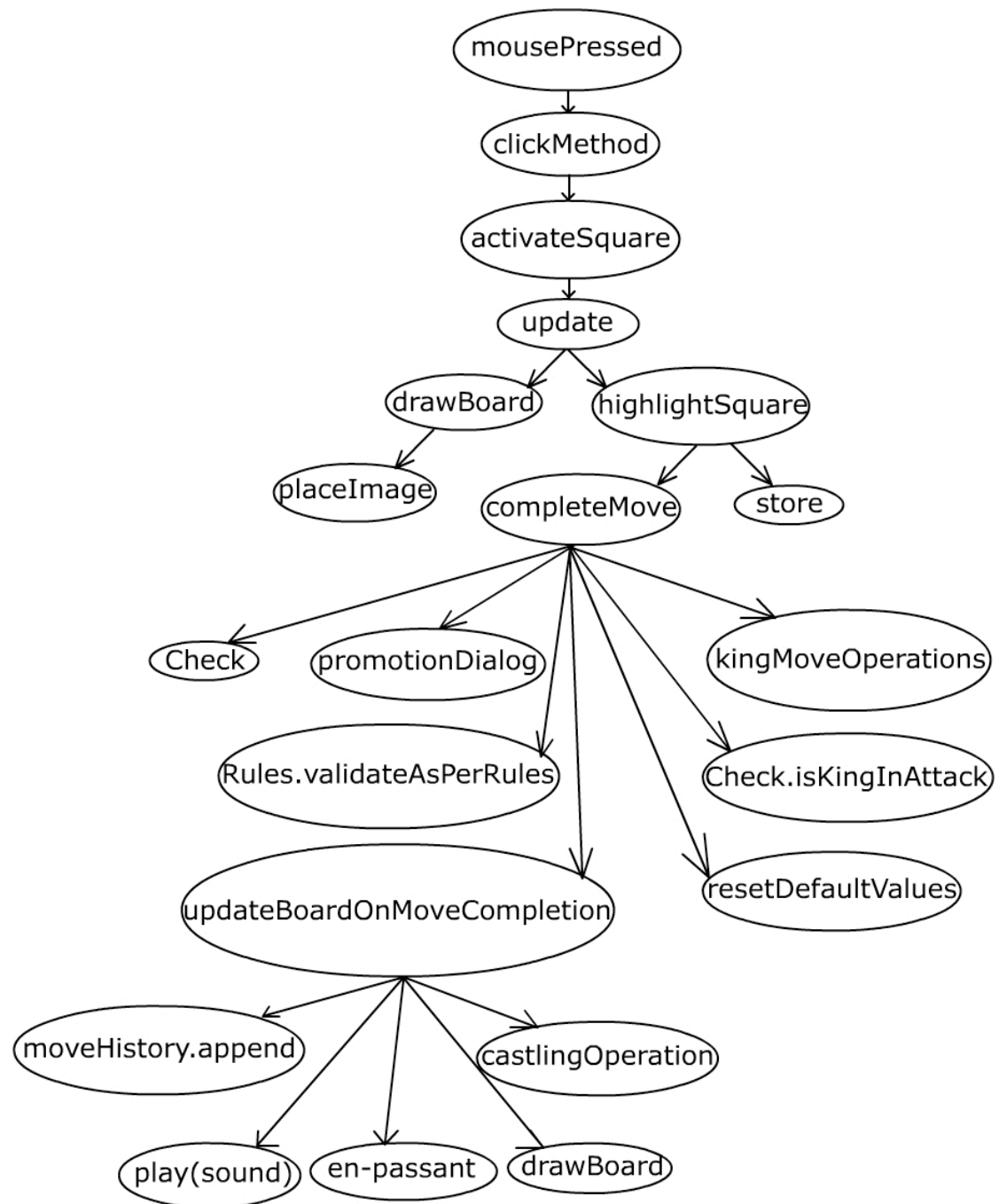
UNDERLYING PROCESS (Client Side)

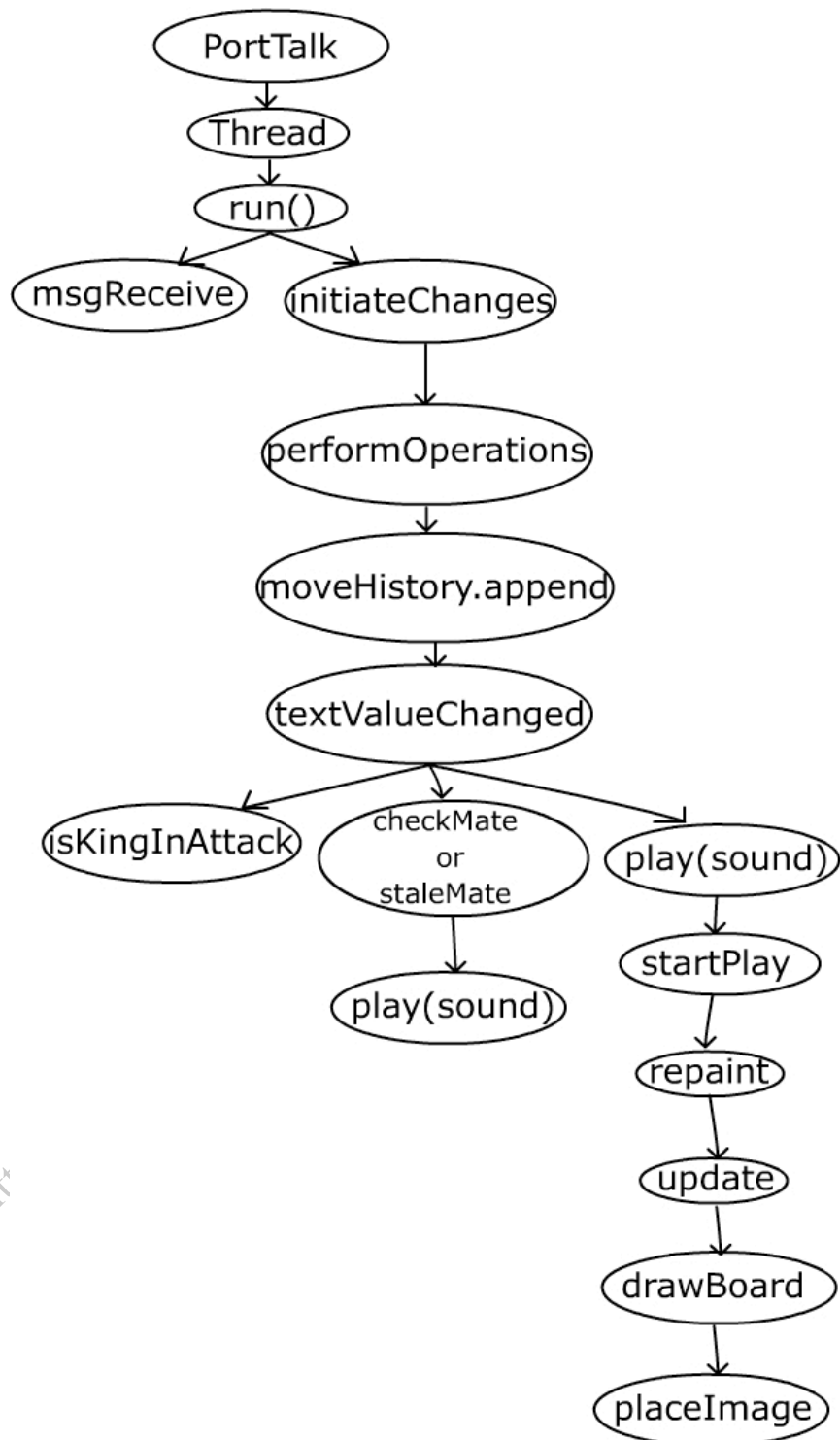
1. Initial Board:



Submitted to Birla Institute of Technology

2. Control Transfer When move made by Current Player:



1. Control transfer to reflect opponent move in the current players Board:

Register:

Two Options:

1. After collecting user information, store information in the database that will make the registration compulsory for any person who want to play the game.
2. Just put user information in the variable like hash table, vector, dictionary. Objects make use of them. So a separate login section is not required.

Based on registration either:

- a. We can go for search in database. If found then continue else go back for registration.
- b. We can go to play game. The server will search for the opponent from the available members in the variables who are not playing yet. When a user (client) login for the game, the server assigns him a port and waits until a new client logs in and then server makes the pair among whom the game is to be played. If one opponent logs out or leave the game, the game gets over.

Internal working of server:

- a. Server is started and waits for the connection request from client at the predefined port
- b. Connection request of client is accepted by the server using ServerSocket and the reference is transferred to the variable of socket class and ServerSocket goes back to listen other Client.
- c. When a new client request is accepted some instance are created i.e. client, his status and his opponent if there is someone. As the info about the client is required to be shared with the opponent we have made these as class variables that is static. So that information can be accessed in between objects. For reading and writing on socket buffered reader and PrintWriter object are created. Opponent is searched from the available list of players and then pairing is done. Messages are transmitted by server to tell the players about their opponents by using server protocol i.e. § is attached by the server for his messages so that the client interpret his message as required. If opponent is not found the player have to wait until an opponent comes into scene.

- d. § is used as string tokenizer as we have assumed this letter will not be used by any user to send information. By using this letter messages are extracted client as well as server end.
- e. After this all initialization work is done we start the thread for client. In this we keep on listening on the thread which is assigned by the server at his location to the client for the communication. When message is received from the client side the message is transmitted to his opponent. If message is for the server like one of the player logging off then server remove the player information form the player details as well as the information of the opponent as a player can't survive alone in the middle when game is ended by the opponent in the current scenario of the game. At this movement of time again the message is transmitted by the server to the opponent that the player has left the game. One needs to restart the game when this kind of situation occurs.
- f. Server thread for each user sleeps in between so that the processing power can be switched between users or clients easily as well as reducing the server load.

Maintaining Player status / opponent info:

In the early days of this project we are just struggling for making the communication establishment in between the no of client. As the time progress we achieved the communication in between the users but now the problem arrived of paring the two players as the game will be played in between two players and no other player group should be disturbed with the moves of another pair playing game. For a while what approach we have taken may be not a good one but fine for now. We have taken three parallel Vectors. This creates a table of $n \times 3$ as shown below, where n is the number of player and 3 is the no of fields maintained for them.

Player1	Status	Player2
V Anand	P	G Kasparov
G Kasparov	P	V Anand
Aditya	P	Vijay
Vijay	P	Aditya
Player New	I	Null

As you can see what we have done here. We are maintaining a list of player with their status and the opponent. If a player is playing his status will be **"P"** (Playing) and the opponent will be any other player in the list. If there is no opponent then the player status will be set to **"I"** (Inactive). On the basis of this table the information is communicated in between player. Instead of names in the table we are storing the socket that holds the info about the port and the ip address of the remote pc.

This table is the key of our whole application which makes the coordination possible like the base chess board at client side. Remote players are updated on the basis of this table only. The type of information is send by using specific protocol for specific type of message. For example if server sends some message the message is having suffix §Server with the type of message it is containing. Like opponent, place black, place white, promotion, castling, logoff and other messages generated by the server.

Similarly other type of messages are transmitted for identifying whether it is board move or message is sent by the opponent form the chat box appended by §Board and §Send respectively

When the game is ended or one player left the game second player is automatically deleted from the list and the message is send before disconnection. One needs to reconnect to access the game.

Displaying the opponent name on the player screen:

This was a problem that appeared in front of us that a player must know who is in opposition. The problem comes when a player is sitting ideal and waiting for some other person. Now when a new player is arrived he is notified with the name of person of opponent easily as his information is already available but the new player info is needed to be provided to the player sitting ideal. For this we created a new protocol that is §opponent which is transmitted by the new player as he connect to server that is the way opponent identify their opponent

6.4 Details of Classes and Methods in the Project:

1) ChessBoard

public class **ChessBoard** extends Applet implements implements
ActionListener, TextListener, MouseListener, MouseMotionListener,
Constants

ChessBoard: Field Summary	
static byte[][]	Board This is a 8X8 Byte Array, for storing the Board Position.
int	c This variable is used to store the column of Board selected by player.
static Boolean	castlingDone This variables is used for castling purpose.
static Boolean	CHECK This variable keeps track whether the current players King is under attack or not.
static Boolean	CHECKMATE This variable keeps track whether the current players is Checkmated or not.
int	col This variable is used to control the highlighting of the squares of the selected by player.
static Boolean	enpassantDone This variable keeps track whether opponent performed en-passant or not.
static Boolean	isKingFresh This variables is used for castling purpose.
static Boolean	isLeftRookFresh This variables is used for castling purpose.
static Boolean	isRightRookFresh This variables is used for castling purpose.
int	oc This variable is used to store the column of Board selected by player.

int	ocol This variable is used to control the highlighting of the squares of the selected by player.
static Boolean	opponentMoved This variable keeps track whether the opponent has moved.
java.lang.String	OpponentName This variable stores the Opponent name.
int	or This variable is used to store the row of Board selected by player.
int	orow This variable is used to control the highlighting of the squares of the selected by player.
static byte	peicepromotedto This variable keeps track of the piece to which the opponent has promoted his pawn.
byte	piecetomove This variable is used to store the code of the peice selected to move by the player.
static Boolean	PlayerColor This variable is used to store the PeiceColor of the Player.
static java.lang.String	previousMoveOFOpponent This variable keeps track of the previous move of the opponent.
int	r This variable is used to store the row of Board selected by player.
int	row This variable is used to control the highlighting of the squares of the selected by player.
static Boolean	STALEMATE This variable keeps track whether the current players is Stalemated or not.
static Boolean	startPlay This variable controls the turn of the players.

ChessBoard: Method Summary	
void	actionPerformed (java.awt.event.ActionEvent ae) Method overridden for implementing ActionListener - Operation is to get the playerName at time of Login AND to send message from the Chat Text Box.
void	activateSquare (int x, int y) This method keeps track of rows and columns variable to highlight the square.
void	castlingOperation (java.lang.String moveDetail) It performs the castling operations.
void	clickMethod (boolean cl, int a, int b) This method keeps track of click variables.
void	completeMove (java.awt.Graphics g) Checks the type of piece and validates the move
void	destroy () This method disconnects the Current Player from the Server.
void	displaySquare (java.lang.String sqsel) Displays the Square which is clicked by user
void	drawBoard (java.awt.Graphics g) Draws the Chess Squares i.e Black and White Square.
void	highlightSquare (java.awt.Graphics g) Draws Red Square around the Square that is active during firstClick It invokes completeMove method during secondClick.
void	init () This method is called only once after applet gets loaded.
void	kingMoveOperation (java.lang.String moveDetail) Performs the King move Operations.
void	loadImages () Loads Images and stores in 2X6 Array of Images with name Pieces.
void	mouseClicked (java.awt.event.MouseEvent me) methods overridden since interface MouseListener implemented.
void	mouseDragged (java.awt.event.MouseEvent me) methods overridden since interface MouseListener implemented.
void	mouseEntered (java.awt.event.MouseEvent me) methods overridden since interface MouseListener implemented.
void	mouseExited (java.awt.event.MouseEvent me) methods overridden since interface MouseListener implemented.
void	mouseMoved (java.awt.event.MouseEvent me) It displays the x,y co-ordinates on Status Bar when mouse is moved.
void	mousePressed (java.awt.event.MouseEvent me) It is the methods which is initiates the start of the move by the player.
void	mouseReleased (java.awt.event.MouseEvent me) methods overridden since interface MouseListener implemented.
void	paint (java.awt.Graphics g)

	method defined to redraw the Applet after each modification.
void	placeImage (java.awt.Graphics g, int i, int j) It draws the Image of piece on the Board.
void	postValidateLogin () Performs one time setup operation.
void	resetDefaultValues () It Resets the row and columns variables.
void	sendMessage (java.lang.String str) Method to concatenate localhost name to String and also Append it with \$Send
void	store (int r, int c) It Stores the or and oc of previous moves and also the piecetomove
void	textChanged (java.awt.event.TextEvent te) Performs the operation when text in MoveHistory is changed
void	textValueChanged (java.awt.event.TextEvent te) Performs operations when MoveHistory is changed This method reflects the Move of the Opponent in the Current Players Board.
void	update (java.awt.Graphics g) This method performs all Graphic related update job It also invokes drawBoard and highlightSquare methods.
void	updateBoardOnMoveCompletion (java.awt.Graphics g, byte piecetomove, java.lang.String moveDetail) It redraws the ChessBoard and updates positions.
void	validateLogin (java.lang.String str) Displays the Login Panel.

-- End of Field and Method Summary of ChessBoard --

2) StartBoard

class **StartBoard** implements Constants

This Class brings the ChessBoard at its starting initial position. ChessBoard is made of 8X8 Byte Array. With bottom most row of the Board from left to right indicating the subscript values from [0][0] to [0][7]. The top most row of the Board from left to right indicating the subscript values from [7][0] to [7][7].

70	71	72	73	74	75	76	77
60	61	62	63	64	65	66	67
50	51	52	53	54	55	56	57
40	41	42	43	44	45	46	47
30	31	32	33	34	35	36	37
20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17
00	01	02	03	04	05	06	07

Here each Chess Piece is assigned a unique code. There are 12 different Chess Pieces in ChessBoard. Their codes are:

BLACK Piece	WHITE Piece
00 – Bpawn	10 – Wpawn
01 – Brook	11 – Wrook
02 – Bknight	12 – Wknight
03 – BBishop	13 – Wbishop
04 – BQueen	14 – Wqueen
05 – Bking	15 – Wking

StartBoard: Method Summary	
void	setBoard() Brings the Board to its initial position.

-- End of Method Summary of StartBoard --

3) Check:

class **Check** implements Constants

This Class performs the calculation part to see whether Current Players King in under CHECK.

Check: Method Summary	
java.lang.String	getKingPos (byte[][] b) This method find the location of the current players king and returns a String. It takes Board Array as argument and scans the array for King code of current player.
boolean	isKingInAttack (byte[][] b) This method is first of the overloaded method to find whether the current players King is under attack. It takes the Board Array as argument and return TRUE if king is under attack. It first invokes the member function getKingPos to find the location of king of current player. Then it scans for opponent peice in the Board Array which is attacking the current player king. If such a piece is found, it returns TRUE. else it returns FALSE.
boolean	isKingInAttack (byte[][] b, int lor, int loc) This method is the second of the overloaded method to find whether the current players King is under attack. It takes the Board Array along with two int variables as arguments and return TRUE if king is under attack. Unlike the first method it does not invoke the method getKingPos as they are already provided by the calling function. Then it scans for opponent peice in the Board Array which is attacking the current player king. If such a piece is found, it returns TRUE. else it returns FALSE.

-- End of Method Summary of Check --

4) CheckMate:

class **CheckMate** implements Constants

This class performs the calculation to see whether current player is CheckMated. It is instantiated only when the current player is under CHECK.

CheckMate: Constructor Summary

CheckMate (byte[] [] b)

This constructor initiates the calculations for checking whether current player is CheckMated. It takes Board Array as argument and invokes helpFromOtherMembers method. If no help is found it set the static variable CHECKMATE as TRUE (OR) STALEMATE as TRUE.

CheckMate : Method Summary

Boolean	<p>doesThereExistSafeMove (int a, int b)</p> <p>This method Checks whether there exist some safe move for the current players King. It takes two integer variables as arguments and invokes isKingInAttack of the class Check. It returns TRUE if there exist such safe move, else returns FALSE.</p>
Boolean	<p>helpFromOtherMembers ()</p> <p>This method tries to get help from pieces other than King. It generates all possible moves of all the peices of the current player and invokes doesThereExistSafeMove method for each such move. Even if for each move there exist a move in which of king is not under attack return FALSE. i.e no CheckMate.</p>

-- End of Method Summary of CheckMate --

5) getPieceOnSquare

class **getPieceOnSquare**

This class is used to located the content of the Board at any given location.

getPieceOnSquare: Method Summary	
static boolean	isBlackOnSquare (byte[][] b, int x, int y) This static method is used to find whether square at a given location of the Board contains BLACK or not. It takes Board Array as argument along with two integer variables. It returns TRUE if BLACK piece is found, else it returns FALSE.
static boolean	isEmptySquare (byte[][] b, int x, int y) This static method is used to find whether square at a given location of the Board is EMPTY or not. It takes Board Array as argument along with two integer variables. It returns TRUE if EMPTY Square is found, else it returns FALSE.
static boolean	isWhiteOnSquare (byte[][] b, int x, int y) This static method is used to find whether square at a given location of the Board contains WHITE or not. It takes Board Array as argument along with two integer variables. It returns TRUE if WHITE piece is found, else it returns FALSE.

-- End of Method Summary of getPieceOnSquare --

Submitted to Birla Institute of Techn

6) Rules

class **Rules** implements Constants

This Class defines the rules for each Chess Piece.

Rules: Constructor Summary	
Rules (byte[][] b)	This constructor takes Board Array as argument and initializes its LocalBoard.

Rules: Method Summary	
boolean	bishopMove (java.lang.String s) This method checks the validity of a BISHOP Move of current Player. It takes String variable as moveDetail and returns a boolean variable. It has 4 possible cases. (Moving Top-Left, Top-Right, Bottom-Left, Bottom-Right). If the move is a valid one it returns TRUE, else it returns FALSE.
boolean	kingMove (java.lang.String s) This method checks the validity of a KNIGHT Move. It takes String variable as moveDetail and returns a boolean variable. KNIGHT has eight possible moves. If the move is a valid one it returns TRUE, else it returns FALSE.
boolean	knightMove (java.lang.String s) This method checks the validity of a KING Move of current Player. It takes String variable as moveDetail and returns a boolean variable. It has three special cases. (Normal Move in 8 directions, Left-Castling and Right-Castling). If the move is a valid one it returns TRUE, else it returns FALSE.
boolean	pawnMoveB (java.lang.String s) This method checks the validity of a PAWN move of Opponent. It takes String variable as moveDetail and returns a boolean variable. It has 5 possible cases. (Move, Two-Square Move, Attack, Promotion and en-passant) If the move is a valid one it returns TRUE, else it returns FALSE.
boolean	pawnMoveW (java.lang.String s) This method checks the validity of a PAWN move of current Player. It takes String variable as moveDetail and returns a boolean variable. It has 5 possible cases. (Move, Two-Square Move, Attack, Promotion and en-passant) If the move is a valid one it returns TRUE, else it returns FALSE.
boolean	queenMove (java.lang.String s) This method checks the validity of a QUEEN Move of current Player. It takes String variable as moveDetail and returns a boolean variable. It combines the Move Logic of ROOK and BISHOP and performs an OR Operation. If the move is a valid one it returns TRUE, else

	it returns FALSE.
boolean	rookMove (java.lang.String s) This method checks the validity of a ROOK Move of current Player. It takes String variable as moveDetail and returns a boolean variable. It has 4 possible cases. (Moving Top, Right, Left, Bottom). If the move is a valid one it returns TRUE, else it returns FALSE.

-- End of Method Summary of Rules --

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

7) PortTalk

class **PortTalk** extends Thread implements Constants

This Class performs the operations of establishing and maintaining client connection with Server. It sends and receives message to and from the Server

PortTalk: Constructor Summary

PortTalk(java.lang.String[] args, java.lang.String playerName)

This constructor establishes connection with the Server at specified Port and Address and initializes various input and output stream variables. It takes two arguments, first one being an array of String containing the Address and Port, second being String of playerName. Once the Connection is established it starts the Thread and keeps watch on the input and output streams.

PortTalk: Method Summary

void	displayDestinationParameters () This method displays the Destination Parameters i.e the Information of the Server. It invokes the displayParameters method.
void	displayLocalParameters () This method displays the Local Parameters i.e the Information of the Player. It invokes the displayParameters method.
void	displayParameters (java.lang.String s, java.lang.String name, byte[] ipAddress, int port) It displays the formatted String passed as input. It receives four arguments, String Host, String Name, byte array of ipAddress and int Port.
void	error (java.lang.String s) This method is invoked whenever an error is accepted. It takes a String as argument. (The Error Message)
java.lang.String	getLocalname () This method is used to return localname as String to the calling method.
java.lang.String	getOnlyMsg (java.lang.String s) It extracts the only the Message from String by dividing the String using field separators.
java.lang.String	getSourceOfMsg (java.lang.String s) It extracts the Source of Message from the String by dividing the String using field separators.
void	initiateChanges (java.lang.String msg)

	<p>This method initiates the Changes when message is received. It takes String variable as argument, which contains a special field separators. It is invoked by the run method.</p>
java.lang.String	<p>msgReceive()</p> <p>This method performs the reading of String from inStream variable. It is invoked by the run method.</p>
void	<p>performOperations(java.lang.String m, java.lang.String s)</p> <p>This method performs various operations on the basis of the Source of Message. It takes two String variables as arguments, one the Message and other the Source. Once the Source is known it performs operations accordingly; like Send, Server, LogOff, RCastling, LCastling, Promotion etc.</p>
void	<p>run()</p> <p>This is invoked when start method is called by PortTalk constructor. It performs runs an indefinite loop to receive and send messages until the player disconnects. It invokes msgReceive method and initiateChanges method.</p>
void	<p>sendData(java.lang.String sendLine)</p> <p>This method sends messages to the Server. It takes a String variable as an argument.</p>
void	<p>shutdown()</p> <p>This method performs the shutdown operation.</p>

-- End of Method Summary of PortTalk --

8) promotionDialog:

class **promotionDialog** implements java.awt.event.ActionListener, java.awt.event.MouseListener, java.awt.event.MouseMotionListener

This class performs the operations of formatting of Dialog Box displayed at the time of Pawn Promotion. It extend the Dialog class and implements appropriate Listeners.

promotionDialog :Constructor Summary	
promotionDialog (java.awt.Frame CB, java.lang.String title)	The constructor passes parameters to the super class constructor i.e Dialog. It accepts two argument first is the Frame object and second is a String for title.

promotionDialog : Method Summary	
void	actionPerformed (java.awt.event.ActionEvent ae) It keeps track peice clicked by the current Player.
void	mouseClicked (java.awt.event.MouseEvent me) This method is Overridden since MouseListener is implemented.
void	mouseDragged (java.awt.event.MouseEvent me) This method is Overridden since MouseListener is implemented.
void	mouseEntered (java.awt.event.MouseEvent me) This method is Overridden since MouseListener is implemented.
void	mouseExited (java.awt.event.MouseEvent me) This method is Overridden since MouseListener is implemented.
void	mouseMoved (java.awt.event.MouseEvent me) This method is Overridden since MouseListener is implemented.
void	mousePressed (java.awt.event.MouseEvent me) This method is Overridden since MouseListener is implemented. It performs the Operation on mousePressed. It assigns the code of the piece clicked by the user and disposes the Dialog window.
void	mouseReleased (java.awt.event.MouseEvent me) This method is Overridden since MouseListener is implemented.
void	paint (java.awt.Graphics g) It performs the paint operations It accept a Graphics Object as argument. It draws, fill rectangles and drawImage.

-- End of Method Summary of promotionDialog --

9) ReverServerApp

public class **ReverServerApp** extends java.lang.Object

The purpose of this class is to receive the connection from the client and then deligate the work to the thread for further processing.

ReverServerApp: Method Summary	
static void	main (java.lang.String[] args) This method is the main method which starts running as the server is started. We have choosed port 1234 for listening client requests. A object of ServerSocket is created and a new object of instanceThread class is created and then started. ServerSocket class throws an IOException.

-- End of Method Summary of ReverServerApp --

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jharkhand)

10) instanceThread:

public class **instanceThread** extends java.lang.Thread

This class is the core of server. It do all the processing part of the server. The instance of this class is created by the ReverServerApp rest processing like for accepting conneciton from the client or setting game, its player , thier status, transmitted messages in between the players or broadcasting some special messages to the players. It extenda the thread class as we need to run process for each single user in the game to intract with the game. Some of the member of this class are kept static as they will be required to share information with oth1er running thread.

instanceThread: Method Summary	
java.lang.String	getOnlyMsg (java.lang.String s) It sends the message part of the string received from the client.
java.lang.String	getSourceOfMsg (java.lang.String s) This message is used to break the message received using StringTokenizer class methods.
void	removePlayer (java.net.Socket current) This method removes the player who has left the game as well as the player who is in the * opponent.
void	run () The whole control is with in this method.
boolean	searchOpponent () Search opponent method is used to searcrh the opponent when a player logs in.
void	summary () The summary method is used just for the server status view.

-- End of Method Summary of instanceThread --

11) sendToCI

public class **sendToCI** extends java.lang.Thread

This class is a used to send the information to the information to its destination. It is extened by the Thread Class.

sendToCI: Method Summary	
void	run() Send the message using printwriter method

-- End of Method Summary of instanceThread --

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

6.5. ScreenShots:

ChessBoard



Chat Text-Boxes

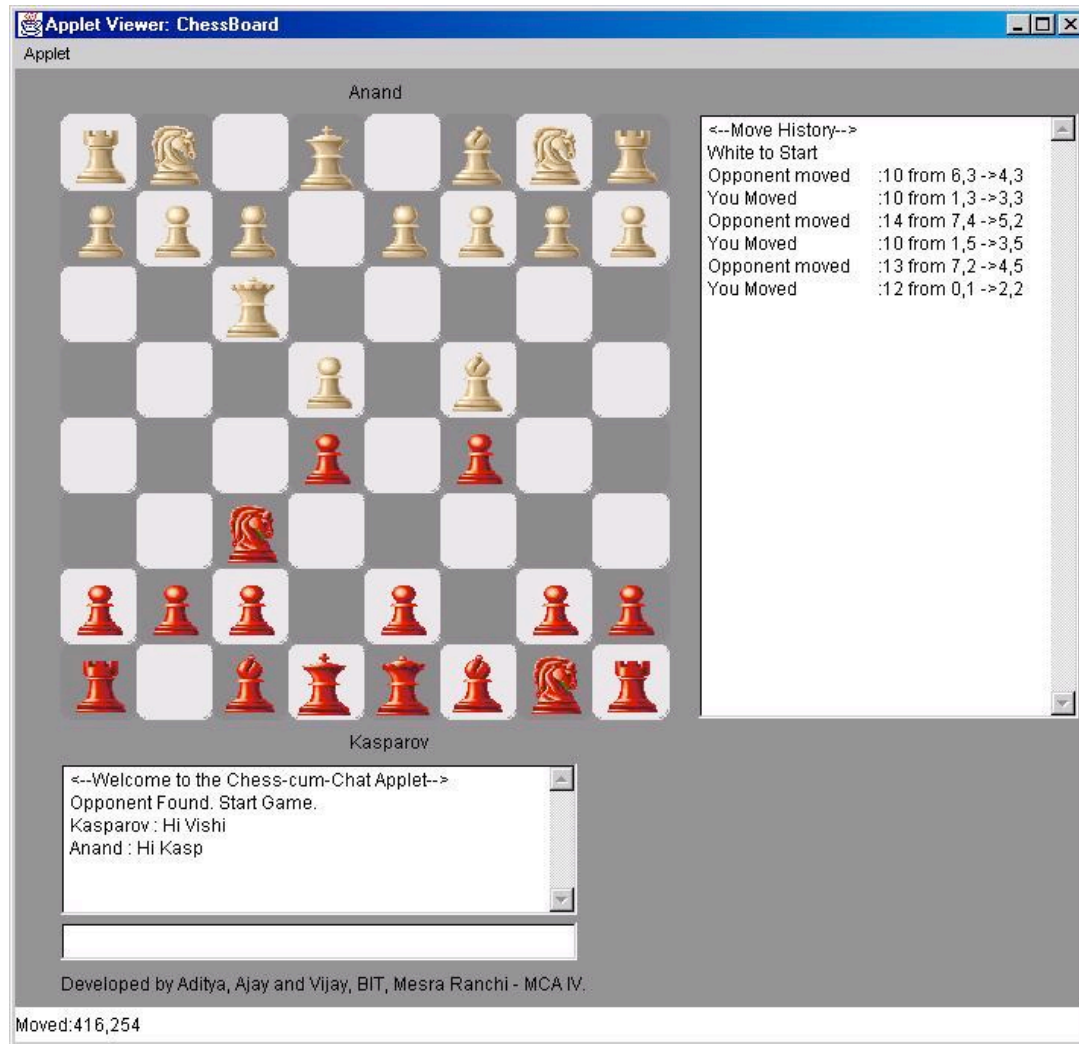


Submitted to Birla Institute of Technology, Mesra, Ranchi

Move-History Box





Complete Applet



Submitted to Birla Inst

6.6. References & sources:

6.6.1 Books:

-  Java Complete Reference.....Herbert Schildt
-  Java Application Development.....STG

6.6.2 Websites:

-  <http://www.instantchess.com/>
(The site inspired us in taking up this project).
-  <http://java.sun.com/>
(Official site of the Sun Microsystems)

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)