INTRODUCTION

PURPOSE OF THE PROJECT

This Project Entitled as 'INTRANET CHATTING' is used basically for chatting purpose with the remote clients or users on Internet or local networks. Here in this project a java client / server combination is used to chat with remote users. When a Client wants to chat with a user on a remote host, he sends a request to the Server with a identification name like chat-id, the server responds to the request by identifying the client-id which is already registered in the server domain and when matched his request is granted and the client can begin to chat with the remote users present on the internet or local network.

The power of Internet is such that it integrates together LANs located across diverse software and hardware forms into a single large communication network that spans the globe. The client needs to have a client software such as Netscape Navigator or Internet Explorer to retrieve information as well as chat on the www. WWW is referred to as Netsurfing. And it supports chatting, text, video and audio.

The benefit of using "INTRANET CHATTING" over other chatting tools is that, with the help of java, the programmer can create applet applications which can be use the internet as a server. Applets are machine independent and so java programs can run on any computer on the internet.

The term client/server is used in the context of networking, what it actually means.

It is important to understand the terms client/server because the INTRANET CHATTTING project is supported completely by client/server model. A server is anything that has some resource that can be shared. There are compute servers, which

provide computing power, web servers, which store web pages. A client is simply any other entity that wants to gain access to a particular server. The interaction between client\server is like an interaction between an electrical socket and a lamp. The server is a permanently available resource while client is free to "unplug" after it has been served.

The program's premier feature is its whiteboard drawing utility. You can draw freehand, do circles, squares, lines, text, or paste image files to the canvas. This is ideal when users want to "sketch" concepts for one another. Users can interact publicly with everyone else on the server, or they can chat/draw privately using java Chat's "whisper mode". Users can create and manage chat rooms, which can be either "public" or "private". The server will also store and manage answering machine-style messages for users who aren't online, and instant messages can be sent to those who are. Additionally, users who aren't watching their screens can be paged with sound.

PROBLEMS EXISTING IN SYSTEM

Have u ever thought that we can chat with people residing At far of places i.e., remote chatting around the world now we can do this, that's what technology is all about and u will be surprised that u can chat with people all over the globe just sitting at your PC, sounds interesting yes!, just at the click of a button u can communicate and share information between different users who are present on their terminals at the time you logged in. All this can be done through a program called 'CHAT' and the project 'INTRANET CHATTING' is all about chatting.

The program is called Chat, when you are Chatting, everything you type will instantly be transmitted around the world to other users that might be watching their terminals at the time. They can then type something and respond to your messages, and vice versa. Chatting is based on a client-server model. Clients are programs that connect to a server; a server is a program that transports data, (messages), from a user client to another. There are clients running on many different systems that allow you to connect to a Chat server. Currently most of the current Chat applications are text based and few are capable of transferring tiny images, but there lacks a Chat system for the technical people who can chat as well as they can represent their ideas in the form of drawing the pictures online.

Many vendors distribute even these technologies separately but to obtain these features at one system will be the haunting task.

So what should we do and how should we solve the problem that even the technical people are benefited by this chatting technology.

Through current chatting technologies we are able to send only text

Based messages to people and tiny images, but this type of chatting is not helping the technical people to work efficiently when the question of sending big pictures like say business plans to the customers that is the business clients to approve of the plan or the client project, imagine big business plans and deals getting approved through chatting and large business projects started, how beneficial it will be to the technical people as well the client on the other side. Let's see how to solve this problem.

SOLUTION TO THE PROBLEM

To solve the inconveniences as mentioned above, an "**INTRANET CHATTING**" is proposed. The proposed system's premier feature is its **whiteboard** drawing utility. You can draw freehand, do circles, squares, lines, text, or paste image files to the canvas. This is ideal when users want to "sketch" concepts for one another. This feature of "INTRANET CHATTING" can be a boon for the technical people who want to share their ideas or concepts in the pictorial form. This system includes the facilities of traditional chat servers and clients like providing a window for each Other user, Whisper chat, multiple chat rooms etc. With the help of the 'WHITE BOARD' drawing utility now the technical people can carry out their tasks easily and can share their big picture plans regarding their business to the clients, exchange ideas and concepts and many more things, basically exchange as well as share the information along with the using the drawing utility even long conversations can be made between two users which may be important business meetings or deals to be sanctioned and all this is carried out with the support of applets with the help of image based web menu images can be transferred.

Scope of the project

WWW is called the World Wide Web. WWW supports many kinds of text, pictures, video and audio. WWW resources through a web browser which basically a program that runs on the internet.

There are two kinds of browsers 1) text only browsers and 2) graphical browsers. Graphical browsers like Netscape Navigator and Internet Explorer are popular. These browsers provide you Inline images, fonts & document layouts. When you access a WWW server, the document is transferred to your computer and then the connection is terminated.

The World Wide Web is a network of information, accessible via an easy-to-use interface. The information is often presented in hypertext or multimedia and provided by servers located around the world. The usability of the Web depends largely on the performance of these servers.

This application is a Java client/server combination, which can be used to chat over the Internet or local networks

With these features and with the advent of WWW, Web browsers and with "INTRANET CHATTING", Internet has become the media of applications.

We can use "INTRANET CHATTING" for following activities:

- To exchange information and converse with friends and family.
- To participate in group discussions through public news bulletin board.
- For Entertainment.
- Leisure activities.
- Access business while at home.
- Communicate and collaborate through pictures and images.
- At any given point of time, up-to-date information is provided.

HARWARE AND SOFWARE SPECIFICATION

- The proposed system should have the following features.
- The Chat Server and Client's Interface should be as simple as possible so that they can be configured even by a naïve user.
- Server should maintain a list of Clients and list of the clients who are currently on line.
- Server should be able to create facility for one to one communication and multiple user communication at once.
- Users can able to share their ideas by drawing.

A user can able to save the chatting information if he feels conversation is important.

PROJECT ANALYSIS

1) STUDY OF THE SYSTEM

This application can be mainly divided into two modules:

1. Server
2. Client

This project "INTRANET CHATTING" is mainly depended on client/server model. The client requests the server and server responses by granting the clients request.

The proposed system should provide both of the above features along with the followed ones:

SERVER: The server should be able to perform the following features:

The first and foremost problem is to find the server. We should identify the program in the server which processes the client's request.

Administrator Client who will be acting as a super user.

Creating of private room with the password facility to enable private chats with the users online. The server is always waiting for clients requests .The clients come and go down but the server remains the same.

CLIENT: The client should be able to perform the following features:

Should be able to send message to anybody in the room with clients unique chat name created in the server for chatting purpose.

Should be provided with the drawing tools like free hand, rectangle, oval, line and also sending text message over the room.

In all the network applications, we find two sort program where the first i.e., server sends the information and the second i.e., client receives the information.

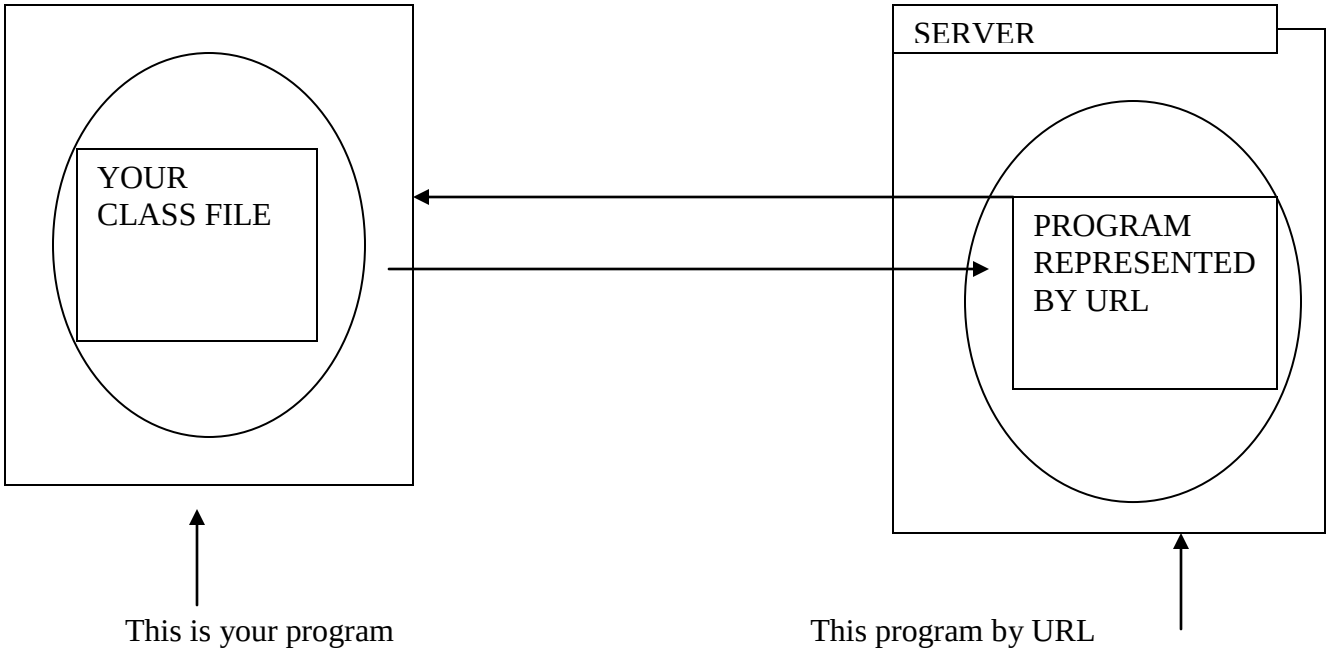
2) INPUT AND OUTPUT:

The user has to provide to which server it has to connect. In this, the server name to be provided is local host.
The user has to provide the username and the password for proceeding with the server for chatting purpose.

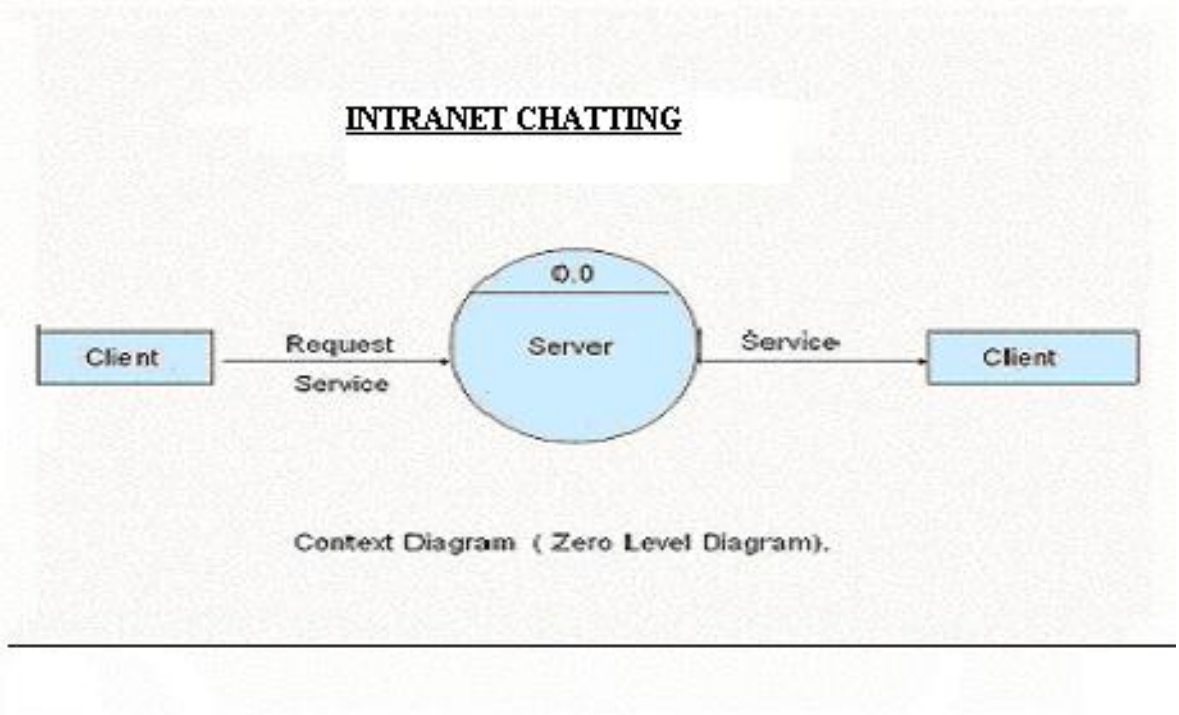
3) PROCESS MODEL USED WITH JUSTIFICATION

The model used here is a SPIRAL MODEL. This Model demands a direct consideration of technical risk at all stages of the project and if properly applied it reduces risk before they become problematic, hence it becomes easier to handle a project when using this kind of model where in the end user can evaluate the program at the end of each stage and suggest modification if required. In this way the Risk Management of Project is carried out efficiently through Spiral model.

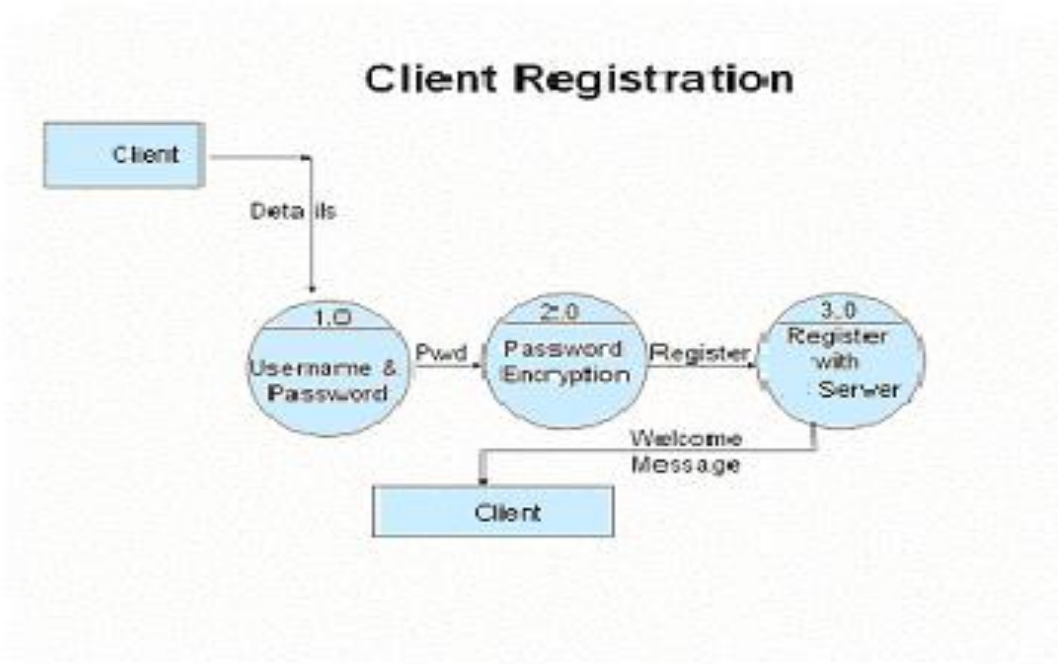
URL CONNECTION



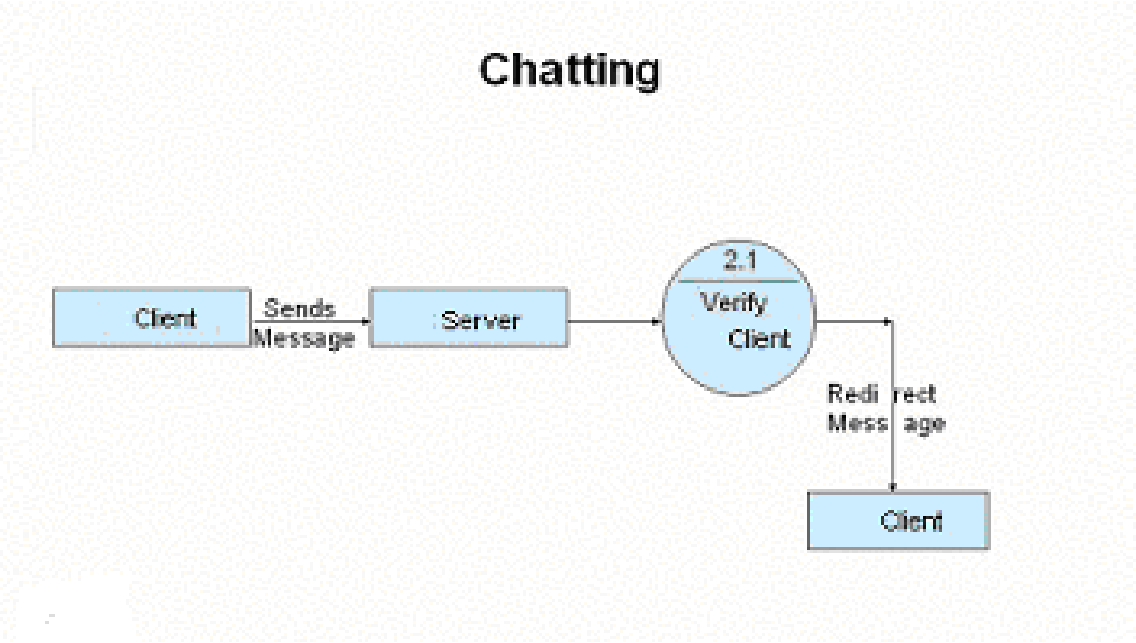
1) DATA FLOW DIAGRAMS

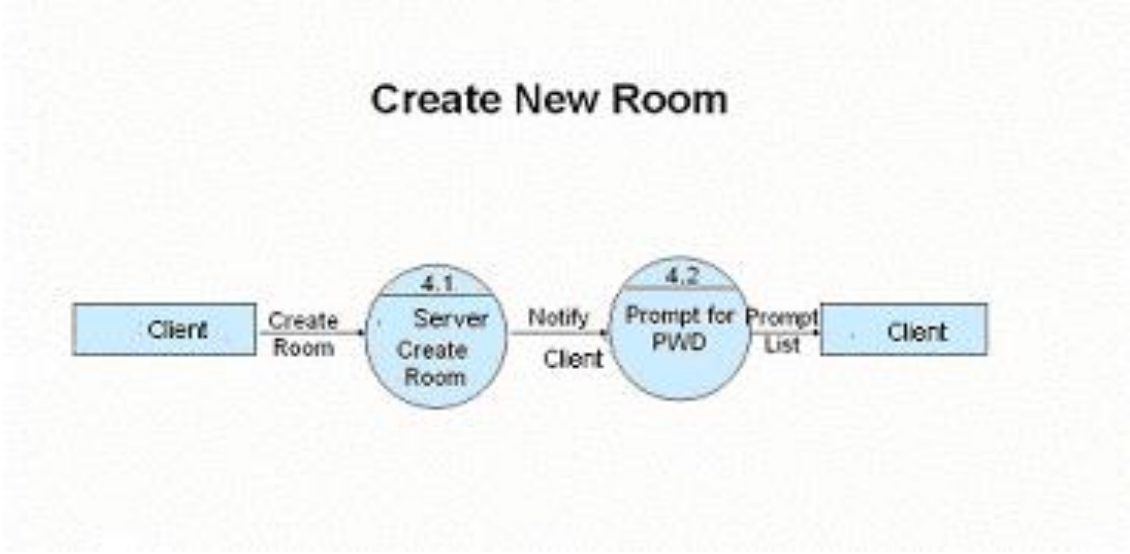


- Client requests service to server and Server grants the request through a response.

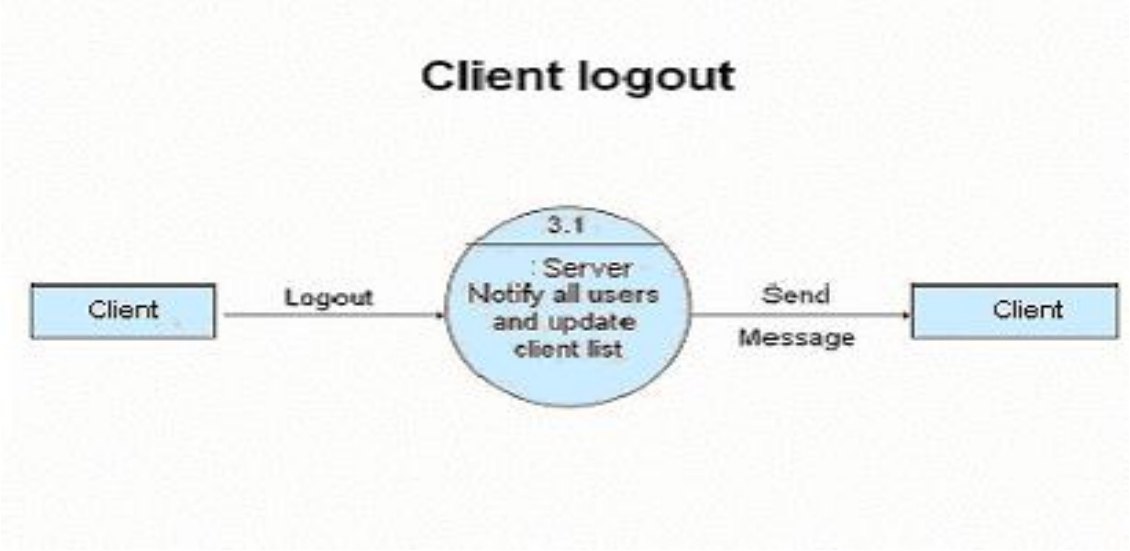


- Client has to first register himself in the server to begin chatting. Server encrypts the PSWD and client is registered, welcome message prompted by server.

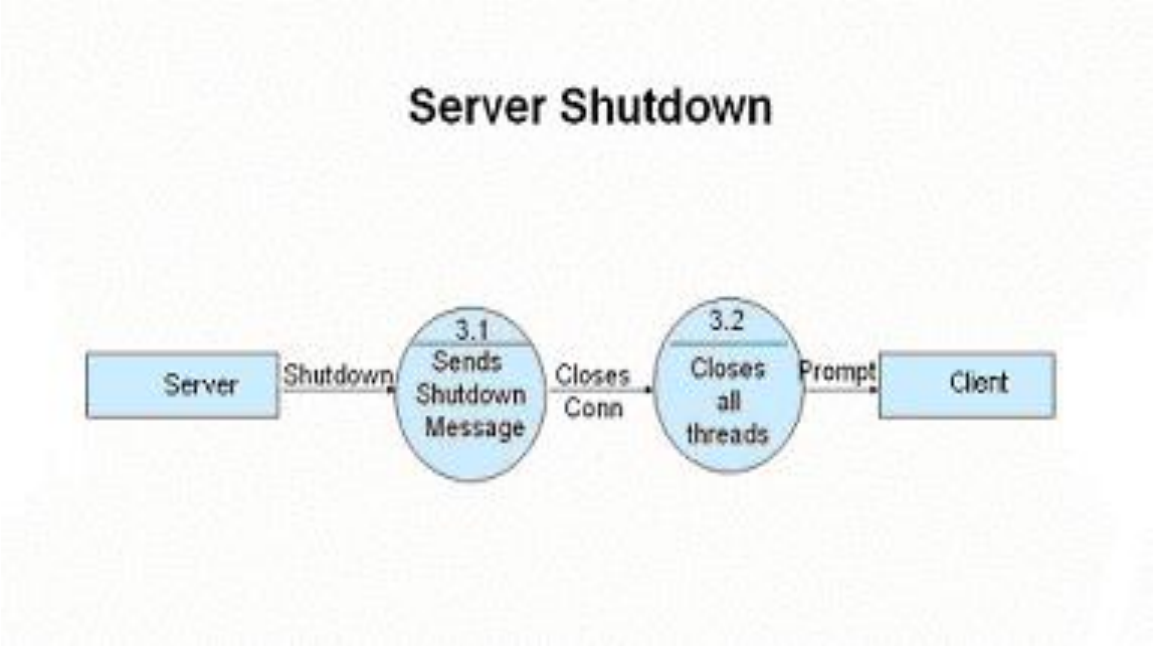




- Client can send offline messages to other clients, server stores and forwards the messages when other user's log on.

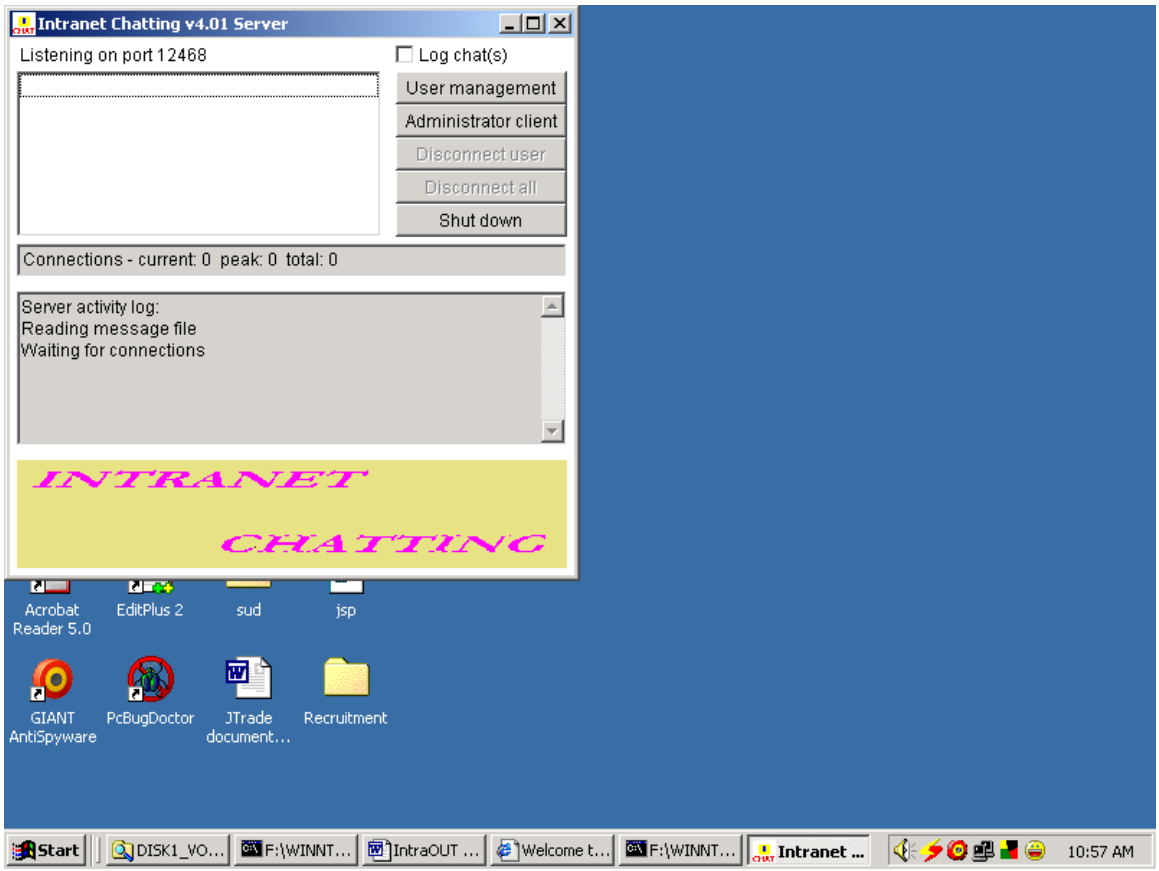


- Client logs out of chat, server notifies & updates all other users by a message.

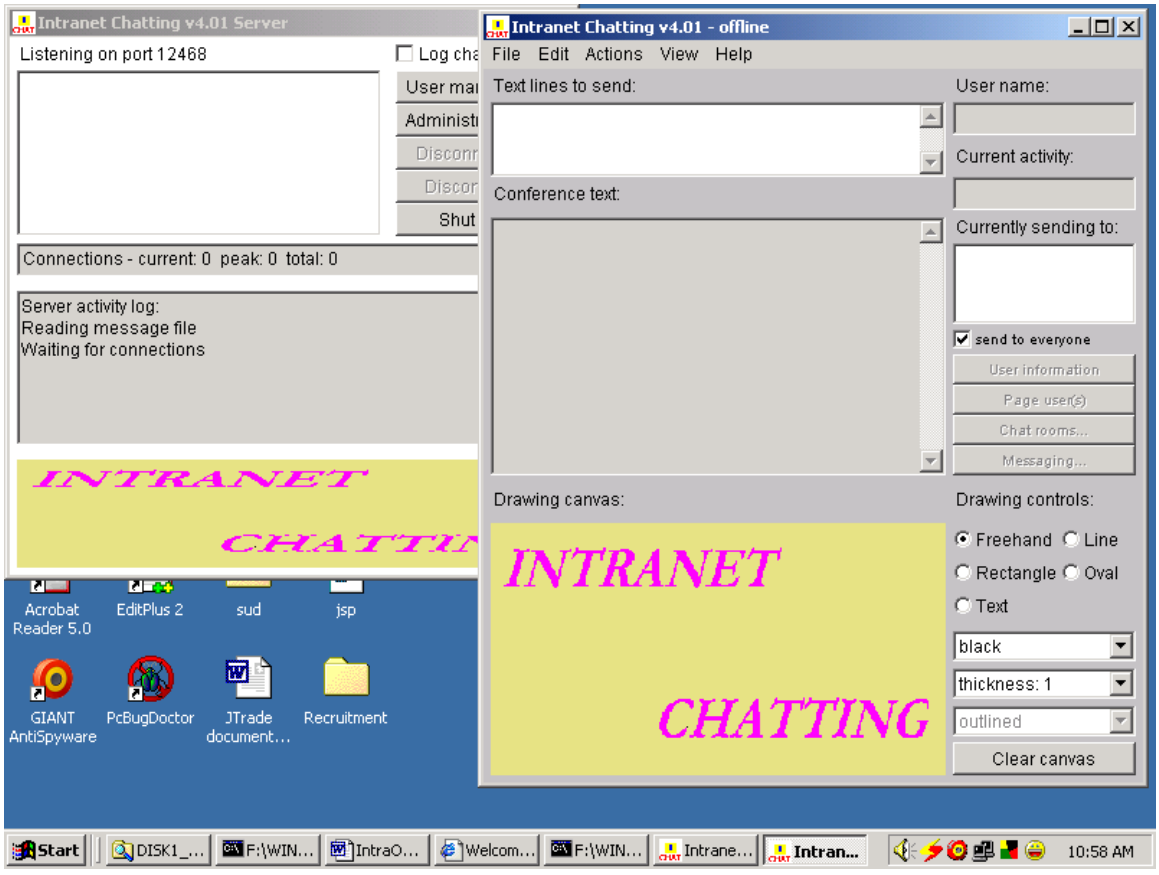


- Server shuts down by sending message and closes connection by giving a prompt.

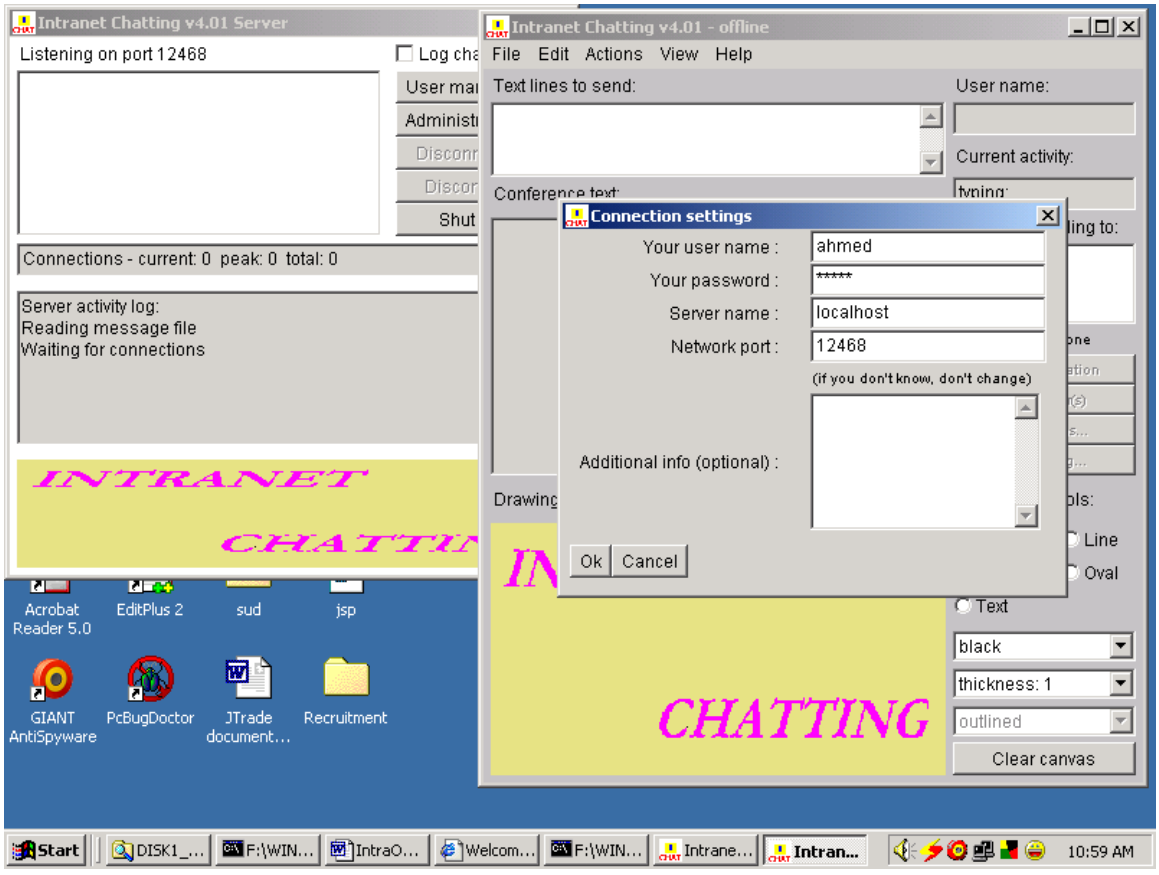
OUTPUT SCREENS



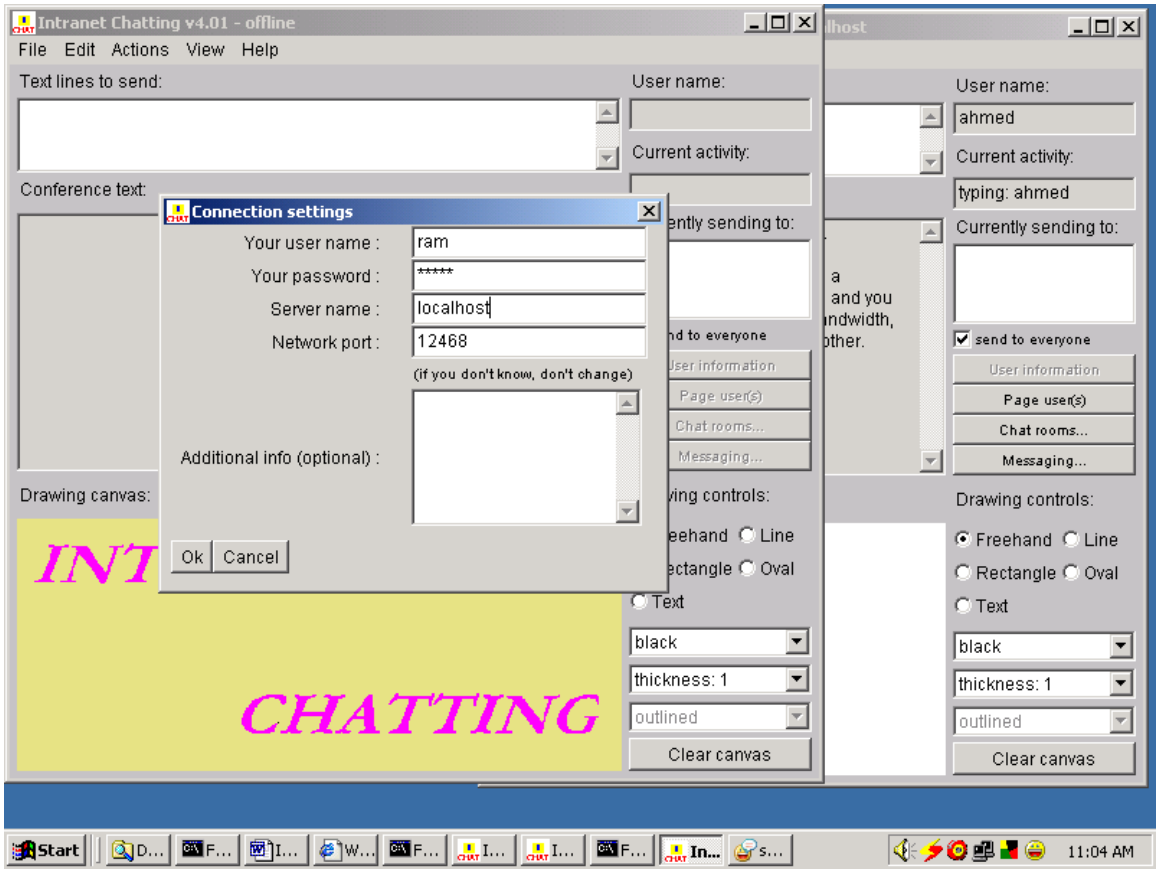
Server Started



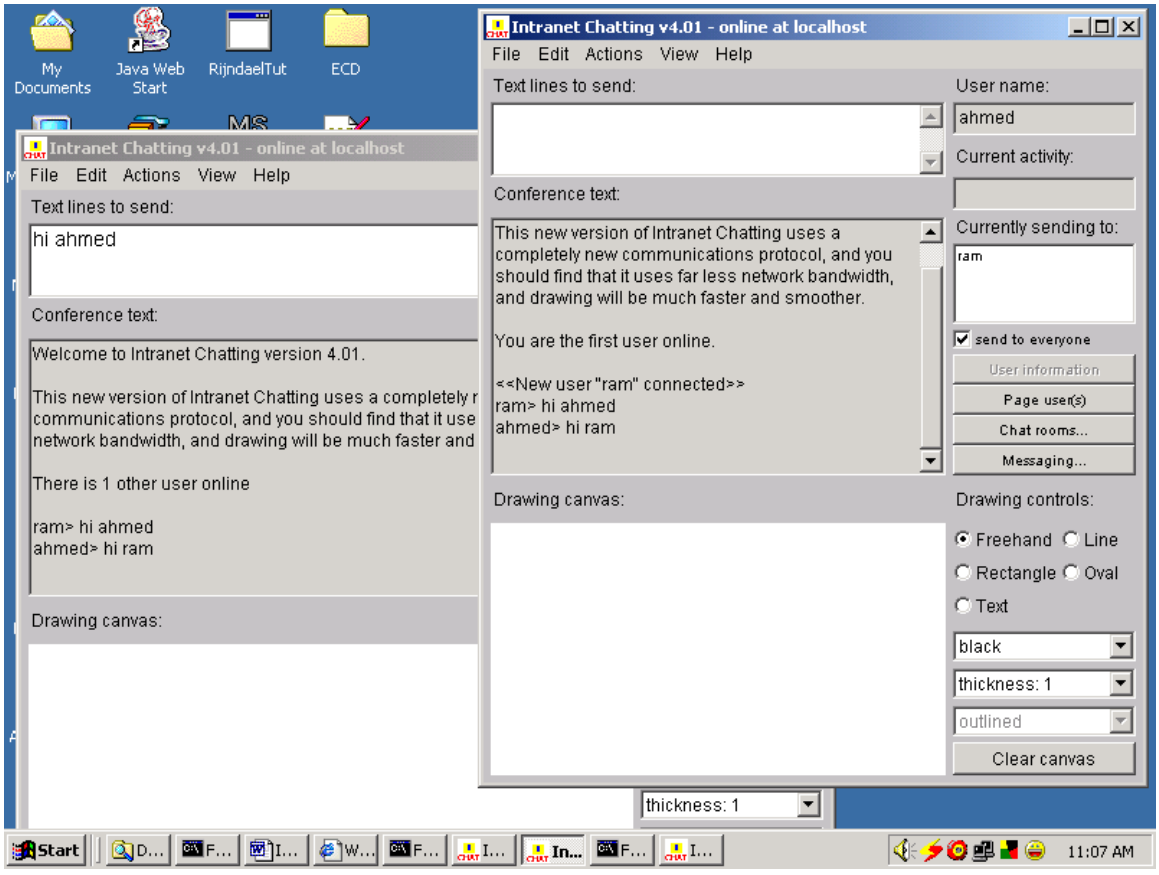
Client Started in another window

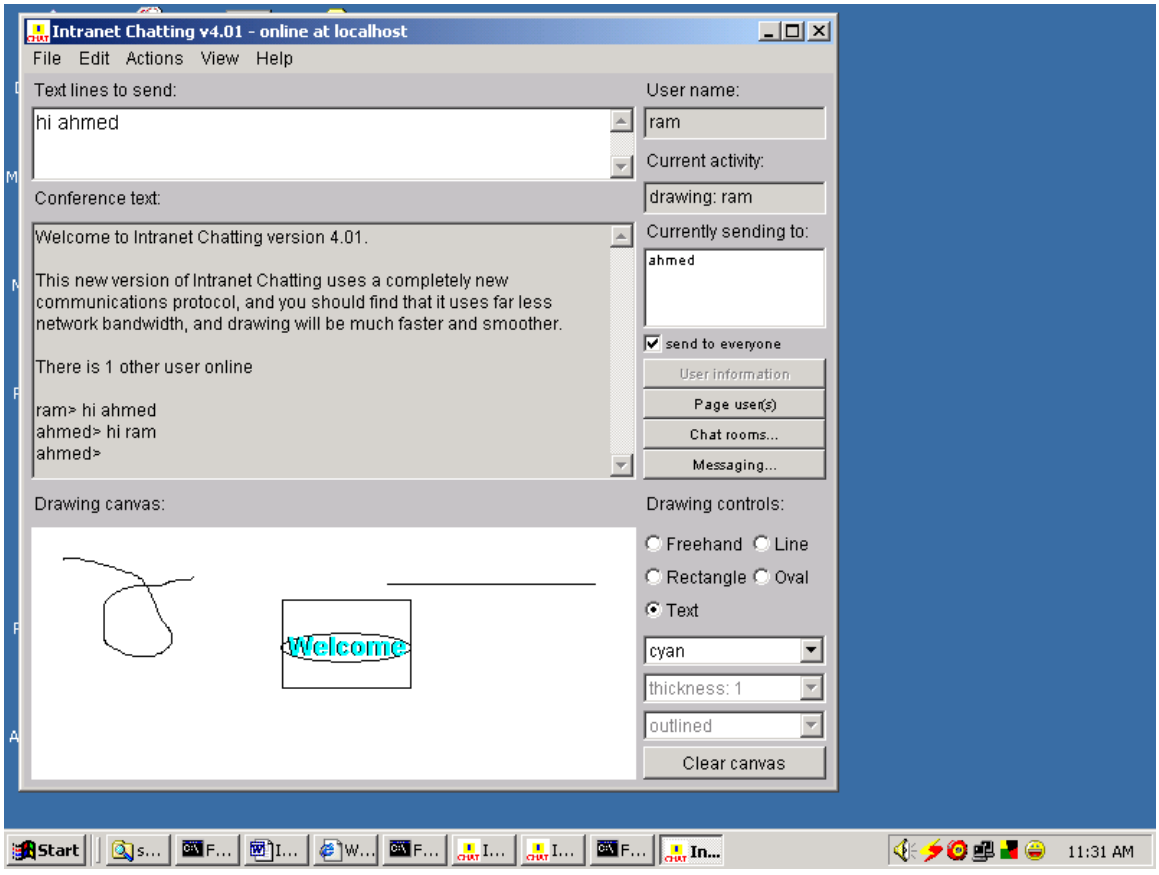


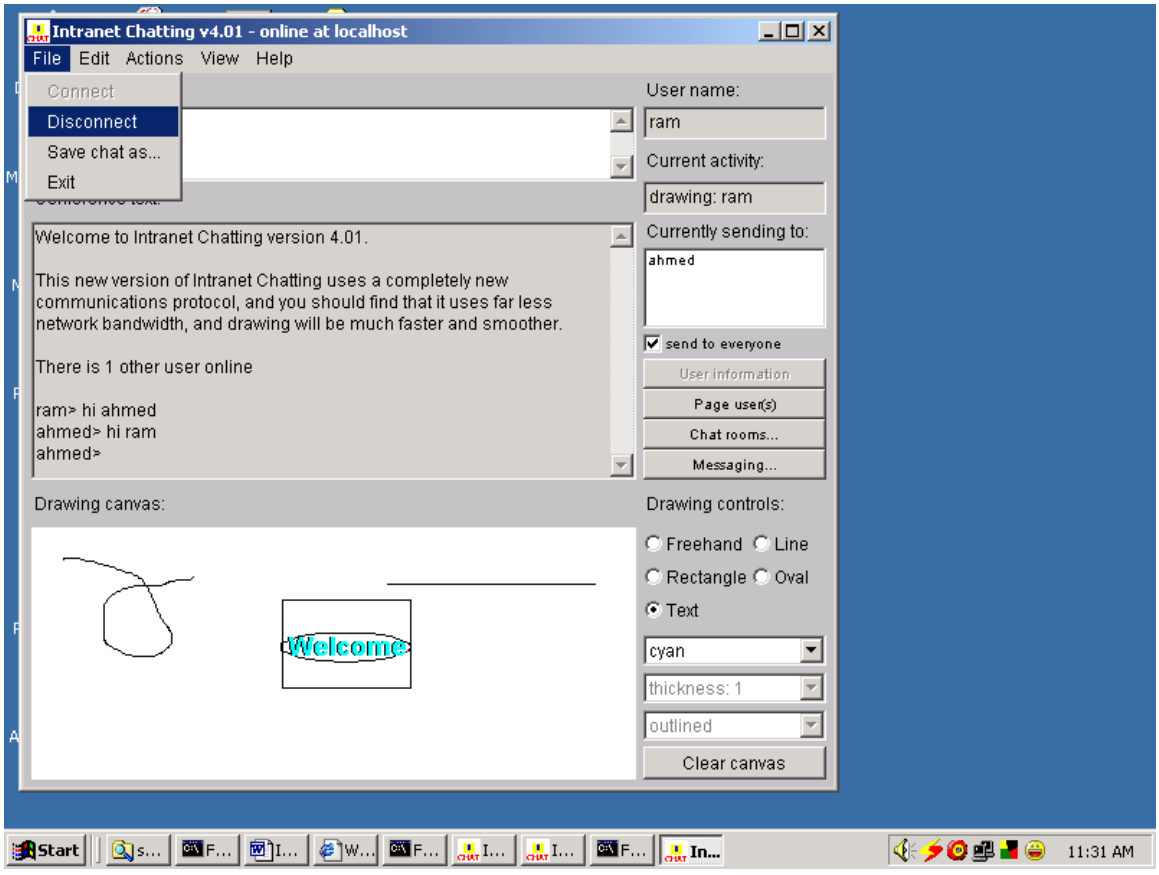
Client Login to Server

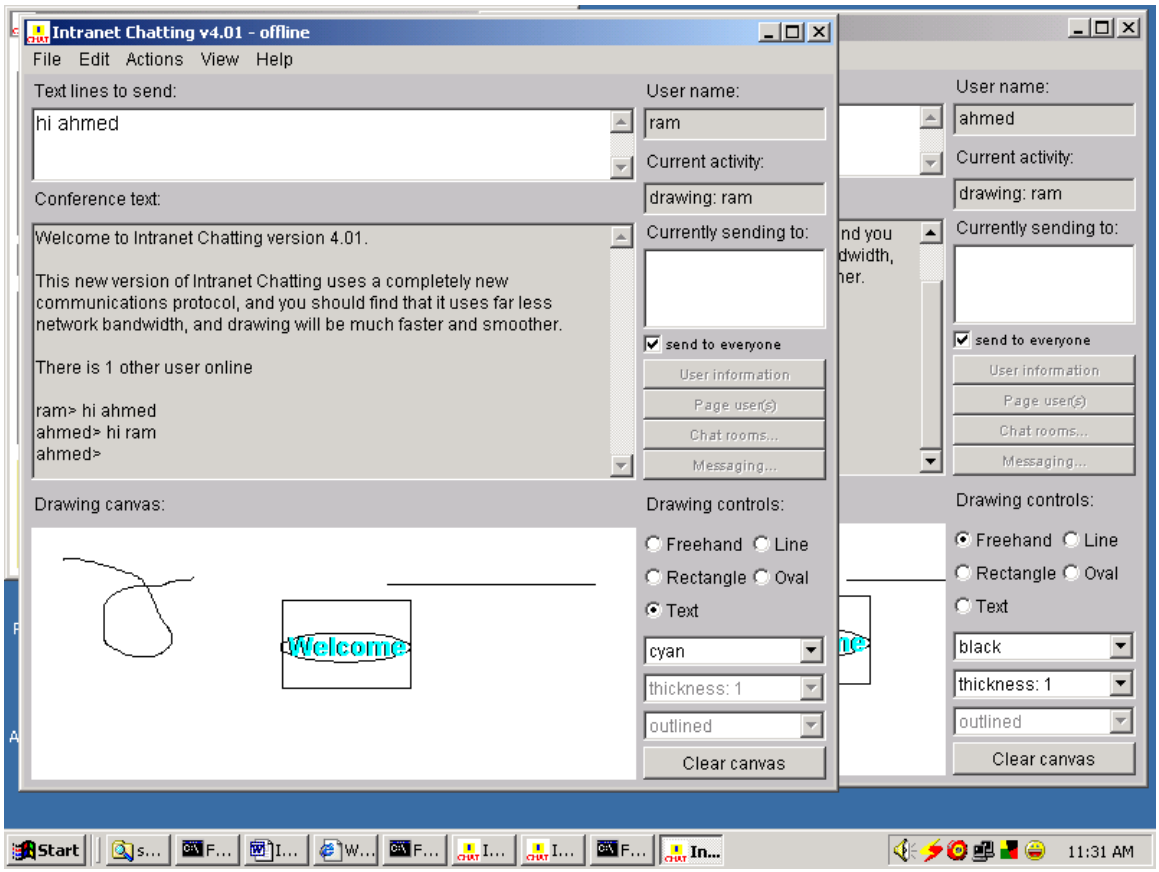

Intranet Chatting v4.01 - offline
File Edit Actions View Help
Text lines to send:
User name:
Current activity:
Conference text:
Connection settings
Your user name :
ram
Your password :
*****
Server name :
localhost
Network port :
12468
(if you don't know, don't change)
Additional info (optional) :
Ok
Cancel
Drawing canvas:
INT
CHATTING
Freehand
Line
Rectangle
Oval
Text
black
thickness: 1
outlined
Clear canvas
User name:
ahmed
Current activity:
typing: ahmed
Currently sending to:
send to everyone
User information
Page user(s)
Chat rooms...
Messaging...
Drawing controls:
Start
11:04 AM

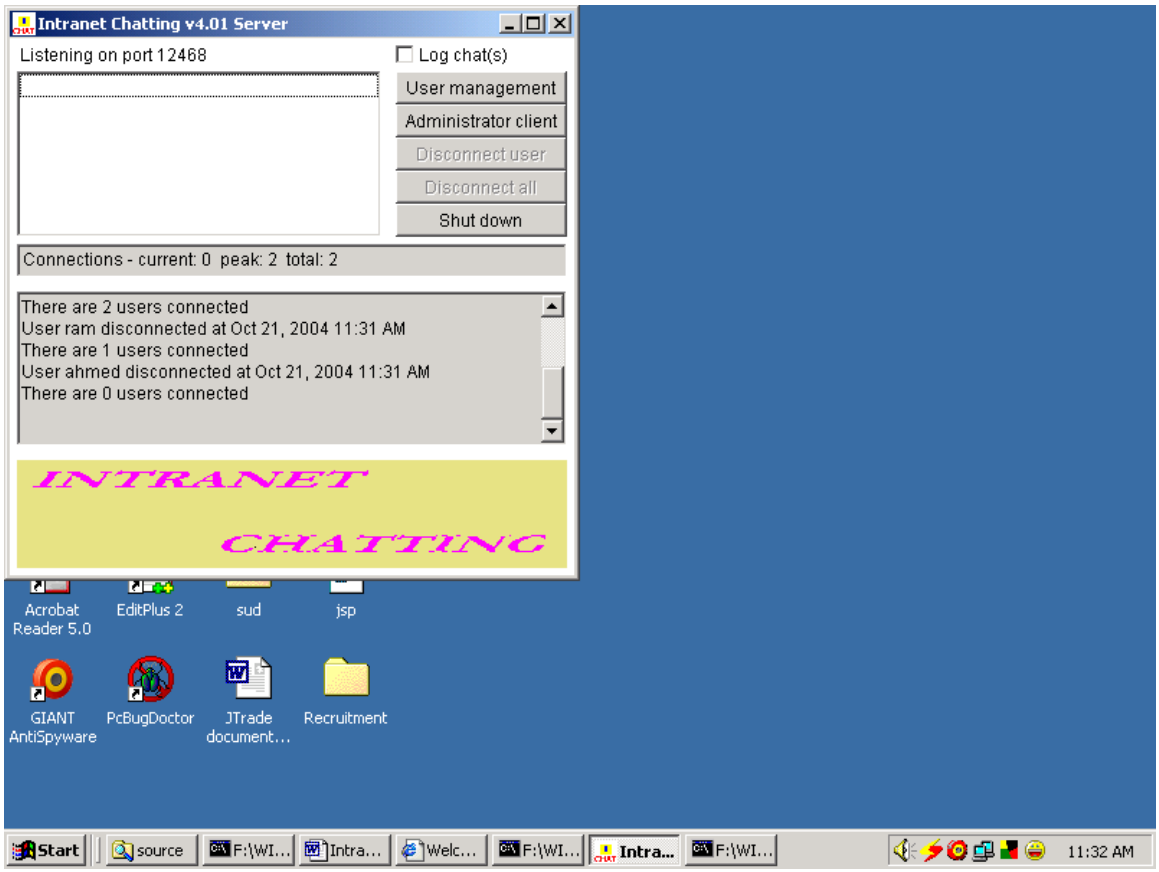
By FaaDoOEngineers.com




Intranet Chatting v4.01 - online at localhost
File Edit Actions View Help
Text lines to send:
hi ahmed
User name:
ram
Current activity:
drawing: ram
Conference text:
Welcome to Intranet Chatting version 4.01.
This new version of Intranet Chatting uses a completely new communications protocol, and you should find that it uses far less network bandwidth, and drawing will be much faster and smoother.
There is 1 other user online
ram> hi ahmed
ahmed> hi ram
ahmed>
Currently sending to:
ahmed
send to everyone
User information
Page user(s)
Chat rooms...
Messaging...
Drawing canvas:
Welcome
Drawing controls:
Freehand Line
Rectangle Oval
Text
cyan
thickness: 1
outlined
Clear canvas
Start
11:31 AM


Intranet Chatting v4.01 - online at localhost
File Edit Actions View Help
Connect
Disconnect
Save chat as...
Exit
User name:
ram
Current activity:
drawing: ram
Currently sending to:
ahmed
send to everyone
User information
Page user(s)
Chat rooms...
Messaging...
Welcome to Intranet Chatting version 4.01.
This new version of Intranet Chatting uses a completely new communications protocol, and you should find that it uses far less network bandwidth, and drawing will be much faster and smoother.
There is 1 other user online
ram> hi ahmed
ahmed> hi ram
ahmed>
Drawing canvas:
Welcome
Drawing controls:
Freehand Line
Rectangle Oval
Text
cyan
thickness: 1
outlined
Clear canvas
Start
11:31 AM




Intranet Chatting v4.01 Server
Listening on port 12468
Log chat(s)
User management
Administrator client
Disconnect user
Disconnect all
Shut down
Connections - current: 0 peak: 2 total: 2
There are 2 users connected
User ram disconnected at Oct 21, 2004 11:31 AM
There are 1 users connected
User ahmed disconnected at Oct 21, 2004 11:31 AM
There are 0 users connected
INTRANET
CHATTING
Acrobat Reader 5.0
EditPlus 2
sud
jsp
GIANT AntiSpyware
PcBugDoctor
JTrade document...
Recruitment
Start
source
F:\WI...
Intra...
Welc...
F:\WI...
Intra...
F:\WI...
11:32 AM

PROJECT CODING

1. CODE EXPLANATION

The INTRANET CHATTING application is developed used awt (Abstract Window Toolkit).

The **java.awt** package is much useful for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components. Some components fire events when a user interacts with the components. A container is a component that can obtain components and other containers. A container can also have a layout manager that controls the visual placement of components in the container.

The **java.awt** package implements different interfaces like **LayoutManager**, which defines the interface for classes that know how to layout Containers.

Paint interface defines how color patterns can be generated for Graphics2D operations. A class implementing the Paint interface is added to the Graphics2D context in order to define the color pattern used by the draw and fill methods.

The **java.net package** provides the classes for implementing networking applications. Using the socket classes, one can communicate with any server on the Internet or implement their own Internet server. A number of classes are

provided to make it convenient to user Universal Resource Locators (URLs) to retrieve data on the Internet.

The java.net package implements different interfaces like **DatagramSocketImplFactory** for implementing data gram socket implementations. Classes DatagramSocket to create actual socket implementation use it.

SocketImplFacotry interface defines a factory for Socket implementations. It is used by the classes socket and ServerSocket to create actual socket implementations.

SocketOptions interface of methods to get/set socket options. Is implemented by SocketImpl and DatagramSocketImpl

OBJECT ORIENTED PROGRAMMING AND JAVA

Object-oriented Programming was developed because of limitations found in earlier approaches of programming. To appreciate what OOP does, we need to understand what these limitations are and how they arose from traditional programming.

PROCEDURAL LANGUAGES

Pascal, C, Basic, FORTRAN, and similar languages are procedural languages. That is, each statement in the language tells the computer to do something: Get some input, add these numbers,, divide by 6, display the output. A program in a procedural language is a list of instructions.

For very small programs no other organizing principle (often called a paradigm) is needed. The programmer creates the list of instructions, and the computer carries them out.

Division into Functions

When programs become larger, a single list of instructions becomes unwieldy. Few programmers can comprehend a program of more than a few hundred statements unless it is broken down into smaller units. For this reason the function was adopted as a way to make programs more comprehensible to their human creators. (The term functions are used in C++ and C. In other languages the same concept may be referred to as a subroutine, a subprogram, or a procedure.) A program is divided into functions, and (ideally, at least) each function has a clearly defined purpose and a clearly defined interface to the other functions in the program. The idea of breaking a program into functions can be further extended by grouping a number of functions together into a larger entity called a module, but the principle is similar: grouping a number of components that carry out specific tasks. Dividing a program into functions and modules is one of the cornerstones of structured programming, the somewhat loosely defined discipline that has influenced programming organization for more than a decade.

Problems with Structured Programming

As programs grow ever larger and more complex, even the structured programming approach begins to show signs of strain. You may have heard about, or been involved in, horror stories of program development. The project is too complex, the schedule slips, more programmers are added, complexity increases, costs skyrocket, the schedule slips further, and disaster ensures. Analyzing the reasons for these failures reveals that there are weaknesses in the procedural paradigm itself. No matter how well the structured programming approach is implemented, large programs become excessively complex.

What are the reasons for this failure of procedural languages? One of the most crucial is the role played by data.

Data Undervalued

In a procedural language, the emphasis is on doing things--read the keyboard, invert the vector, check for errors, and so on. The subdivision of a program into functions continues this emphasis. Functions do things just as single program statements

do. What they do may be more complex or abstract, but the emphasis is still on the action.

What happens to the data in this paradigm? Data is, after all, the reason for a program's existence. The important part of an inventory program isn't a function that displays the data, or a function that checks for correct input; it's the inventory data itself. Yet data is given second-class status in the organization of procedural languages.

For example, in an inventory program, the data that makes up the inventory is probably read from a disk file into memory, where it is treated as a global variable. By global we mean that the variables that constitute the data are declared outside of any function, so they are accessible to all functions. These functions perform various operations on the data. They read it, analyze it, update it, rearrange it, display it, and write it back to the disk, and so on.

We should note that most languages, such as Pascal and C, also support local variables, which are hidden within a single function. But local variables are not useful for important data that must be accessed by many different functions. Now suppose a new programmer is hired to write a function to analyze this inventory data in a certain way.

Unfamiliar with the subtleties of the program, the programmer creates a function that accidentally corrupts the program. This is easy to do, because every function has complete access to the data. It's like leaving your personal papers in the lobby of your apartment building: Anyone can change or destroy them. In the same way, global data can be corrupted by functions that have no business changing it.

Another problem is that, since many functions access the same data, the way the data is stored becomes critical. The arrangement of the data can't be changed without modifying all the functions that access it. If you add new data items, for example, you'll need to modify all the functions that access the data so that they can also access these new items. It will be hard to find all such functions, and even harder to modify all of them correctly. It's similar to what happens when your local supermarket moves the bread from aisle 4 to

aisle 12. Everyone who patronizes the supermarket must figure out where the bread has gone, and adjust their shopping habits accordingly.

What is needed is a way to restrict access to the data, to hide it from all but a few critical functions. This will protect the data, simplify maintenance, and offer other benefits as well.

Relationship to the Real World

Procedural programs are often difficult to design. The problem is that their chief components--functions and data structures--don't model the real world very well. For example, suppose you are writing a program to create the elements of a graphics user interface: menus, windows, and so on. Quick now, what functions will you need? What data structures? The answers are not obvious, to say the least. It would be better if windows and menus corresponded more closely to actual program elements.

New Data Types

There are other problems with traditional languages. One is the difficulty of creating new data types. Computer languages typically have several built-in data types: integers, floating-point numbers, characters, and so on. What if you want to invent your own data type? Perhaps you want to work with complex numbers, or two dimensional coordinates, or dates—quantities the built-in data types don't handle easily. Being able to create your own types is called extensibility; you can extend the capabilities of the language. Traditional languages are not usually extensible. Without unnatural convolutions, you can't bundle together both X and Y coordinates into a single variable called Point, and then add and subtract values of this type. The result is that traditional programs are more complex to write and maintain.

The object oriented approach

The fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an object.

An object's functions, called member methods in Java, typically provide the only way to access its data. If you want to read the item and return the value to you, you call a member function in the object. It will read the item and return the value to you. You can't access the data directly. The data is hidden, so it is safe from accidental modification. Data and its functions are said to be encapsulated into a single entity. Data encapsulation and data hiding are key terms in the description of object oriented languages. If you want to modify the data in an object, you know exactly what functions interact with it: the member functions in the object. No other functions can access the data. This simplifies writing, debugging, and maintaining the program.

A Java program typically consists of a number of objects, which communicate with each other by calling one another's members functions. We should mention that what are called member functions in C++ are called methods in Java. Also, data items are referred to as instance variables. Calling an object's member function is referred to as sending a message to the object.

An analogy

You might want to think of objects as departments—such as sales, accounting, personnel, and so on—in a company. Departments provide an important approach to corporate organization. In most companies (except very small ones), people don't work on personnel problems one day, the payroll the next, and then go out

in the field as sales people the week after. Each department has its own personnel, with clearly assigned duties. It also has its own data: payroll, sales figures, personnel records, inventory, or whatever, depending on the department.

The people in each department control and operate on those departments data. Dividing the company into departments makes its easier to comprehend and control the company's activities, and helps them maintain the integrity of the information used by the company. The payroll department, for instance, is responsible for the payroll data. If you are from the sales department, and you need to know the total of all the salaries paid in the southern region in July, you don't just walk into the payroll department and start running through file cabinets. You send a memo to the appropriate person in the department, and then you wait for that person to access the appropriate person in the department, and then you wait for that person to access the data and send you a reply with the information you want. This ensures that the data is accessed accurately and that it is not corrupted by inept outsiders. (This view of corporate organization is show in figure). In the same way, objects provide an approach to program organization, while helping to maintain the integrity of the programs data.

OOP: An approach to organization

Keep in mind that object-oriented programming is not primarily concerned with the details of program operation. Instead, it deals with the overall organization of the program.

Characteristics of object-oriented languages:

Let's briefly examine a few of the major elements of object-oriented languages in general and Java in particular.

Objects

When you approach a programming problem in an object oriented language, you no longer ask how the problem will be divided into functions, but how it will be divided into objects. Thinking in terms of objects, rather than functions, has a surprisingly helpful effect on how easily programs can be designed and objects in the real world.

What kinds of things become objects-oriented programs? The answer to this is limited only by your imagination, but there are some typical categories to start you thinking:

Physical objects

Automobile in a traffic-flow simulation

Electrical components in a circuit design to a program

Countries in an economics model

Aircraft in an air-traffic control system

- *Elements of the computer-user environment*
    - Windows
    - Menus
    - Graphics objects(lines ,rectangles, circles)
    - The mouse and the keyboard

- Programming constructs
    - Customized arrays
    - Stacks
    - Linked lists

- Collection of data
    - An inventory
    - A personnel file
    - A dictionary

  A table of the latitudes and longitudes of world cities
- User defined data types
    - Time
    - Angles
    - Complex numbers
    - Points on the plane

- Components in a computer games

  Ghosts in maze game

  Positions in a board game (chess, checkers)

  Animals in an ecological simulation

  Opponents and friends in adventure games

  The match between programming objects and real-world objects us the happy result of combining data and functions: the resulting objects offer a revolution in program designing, no such close match between programming constructs and the items being modeled exists in a procedural language.

Classes

In OOP we say that objects are members of classes. What does this mean? Let's look at an analogy. Almost all computer languages have built-in data types. For instance, a data type int, meaning integer is pre-defined in Java. You can declare as many variables of type int as you need in your program:

```
Int day;
Int count;
Int divisor;
Int answer;
```

A class serves as a plan, or template. It specifies what data, and what functions will be included in objects of that class. Defining the class doesn't create any objects, just as the mere existence of a type int doesn't create any variables.

A class is thus a collection of similar objects. This fits our non technical understanding of the word class, Prince, sting etc., are members of the class of rock musicians. There is no person called rock musician but specific people with specific names are members of this class if they possess certain characteristics.

Abstraction

An essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. For example, people do not think of a car as a set of tens of thousands of individual parts. They think of it as a well-defined object with its own unique behavior. This abstraction allows people to use a car to drive to the grocery store without being overwhelmed by the complexity of the parts that form the car. They can ignore the details of how the engine, transmission, and braking systems work. Instead they are free to utilize the object as a whole.

A powerful way to manage abstraction is through the use of hierarchical classifications. This allows you to layer the semantics of complex systems, breaking them into more manageable pieces. From the outside, the car is a single object. Once inside, you see that the car consists of several subsystems: steering, brakes, sound system, seat belts, heating, cellular phone, and so on. In turn, each of these subsystems is made up of more specialized units. For instance, the sound system consists of a radio, a CD player, and/or a tape player. The point is that you manage the complexity of the car (or any other complex system) through the use of hierarchical abstractions.

Hierarchical abstractions of complex systems can also be applied to computer programs. The data from a traditional process-oriented program can be transformed by abstraction into its component objects. A sequence of process steps can become a collection of messages between these objects. Thus, each of each object describes its own unique behavior. You can treat these objects as concrete entities that respond to messages telling them to do something. This is the essence of object-oriented programming.

Object-oriented concepts form the heart of Java just as they form the basis for human understanding. It is important that you understand how these concepts translate into programs. As you will see, object-oriented programming is a powerful and natural paradigm for creating programs that survive the inevitable changes accompanying the life cycle of any major software project, including conception, growth, and aging. For

example, once you have a well-defined objects and clean, reliable interfaces to those objects, you can gracefully decommission or replace parts of an older system without fear.

Encapsulation

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper. Access to the code and data inside the wrapper is tightly controlled through a well-defined interface. To relate this to the real world, consider the automatic transmission on an automobile. It encapsulates hundreds of bits of information about your engine, such as how much you are accelerating, the pitch of the surface you are on, and the position of the shift lever. You, as the user, have only one method of affecting this complex encapsulation: by moving the gear-shift lever. You can't affect the transmission by using the turn signal or windshield wipers, for example. Thus, the gear-shift lever is a well-defined (indeed, unique) interface to the transmission. Further, what occurs inside the transmission does not affect objects outside the transmission. For example, shifting gears does not turn on the headlights! Because an automatic transmission is encapsulated, dozens of car manufacturers can implement one in any way they please. However, from the

Driver's point of view, they all work the same. This same idea can be applied to programming. The power of encapsulated code is that everyone knows how to access it and thus can use it regardless of the implementation details—and without fear of unexpected side effects.

In Java the basis of encapsulation is the class. Although the class will be examined in great detail later in this book, the following brief discussion will be helpful now. A class defines the structure and behavior (data and code) that will be shared by a

set of objects. Each object of a given class contains the structure and behavior defined by the class, as if it were stamped out by a mold in the shape of the class. For this reason, objects are sometimes referred to as instances of a class. Thus, a class is a logical construct; an object has physical reality.

When you create a class, you will specify the code and data that constitute that class. Collectively, these elements are called members of the class. Specifically, the data defined by the class are referred to as member variables or instance variables. The code that operates on that data is referred to as member methods or just methods.

Since the purpose of a class is to encapsulate complexity, there are mechanisms for hiding the complexity of the

Implementation inside the class. Each method or variable in a class may be marked private or public. The public interface of a class represents everything that external users of the class need to know, or may know. The private methods and data can only be accessed by code that is a member of the class. Therefore, any other code that is not a member of the class cannot access a private method or variable. Since the private members of a class may only be accessed by other parts of your program through the class' public methods, you can ensure that no improper actions take place. Of course, this means that the public interface should be carefully designed not to expose too much of the inner workings of a class.

Inheritance

Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification. As mentioned earlier, most knowledge is made manageable by hierarchical (that is, top-down) classifications. For example, a Golden Retriever is part of the classification dog, which in turn is part of the mammal class, which is under the larger class animal. Without the use of hierarchies, each object would need to define all of its

Characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes

from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

Most people naturally view the world as made up of objects that are related to each other in a hierarchical way, such as animals, mammals, and dogs. If you wanted to describe animals in an abstract way, you would say they have some attributes, such as size, intelligence, and type of skeletal system. Animals also have certain behavioral aspects; they ear, breathe, and sleep. This description of attributes and behavior is the class definition for animals. If you wanted to describe a more specific class of animals, such as mammals, they would have more specific attributes, such as type of teeth, and mammary glands. This is known as a subclass of animals, where animals are referred to as mammals' super class.

Since mammals are simply more precisely specified animals, they inherit all of the attributes from animals. A deeply inherited subclass inherits all of the attributes from each of its ancestors in the class hierarchy. Inheritance interacts with encapsulation as well. If a given class encapsulates some attributes, then any subclass will have the same attributes plus any that it adds as part of its specialization (see Figure 2-2). This is a key concept which lets object-oriented programs grow in complexity linearly rather than geometrically. A new subclass inherits all of the attributes of all of its ancestors. It does not have unpredictable interactions with the majority of the rest of the code in the system.

Polymorphism

Polymorphism (from the Greek, meaning "many forms") is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. Consider a stack (which is a last-in, first-out list). You might have a program that requires three types of stack. One stack is used for integer values, one for floating-point values, and one for characters. The algorithm that implements each stack is the same, even though the data being stored differs. In a non-object-oriented language, you would be required to create three difference sets of stack routines, with each set using different names. However, because of polymorphism, in Java you can specify a general set of stack routines that all share the same names.

More generally, the concept of polymorphism is often expressed by the phrase "one interface, multiple methods." This means that it is possible to design a generic interface to a group of related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler's job to select the specific action (that is, method) as it applies to each situation. You, the programmer, do not need to make this selection manually. You need only remember and utilize the general interface.

Extending the dog analogy, a dog's sense of smell is polymorphic. If the dog smells a cat, it will bark and run after it. If the dog smells its food, it will salivate and run to its bowl. The same sense of smell is at work in both situations. The difference is what is being smelled, that is, the type of data being operated upon by the dog's nose! This same general concept can be implemented in Java as it applies to methods within a Java program.

Polymorphism, Encapsulation, and Inheritance Work Together

When properly applied, polymorphism, encapsulation, and inheritance combine to produce a programming environment that supports the development of far more robust and scaleable programs than does the process-oriented model. A well-designed hierarchy of classes is the basis for reusing the code in which you have invested time and effort developing and testing. Encapsulation allows you to migrate your implementations over

Time without breaking the code that depends on the public interface of your classes. Polymorphism allows you to create clean, sensible, readable, and resilient code.

Of the two real-world examples, the automobile more completely illustrates the power of object-oriented design. Dogs are fun to think about from an inheritance standpoint, but cars are more like programs. All drivers rely on inheritance to drive different types (subclasses) of vehicles. Whether the vehicle is a school bus, a Mercedes sedan, a Porsche, or the family minivan, drivers can all more or less find and operate the steering wheel, the brakes, and the accelerator. After a bit of gear grinding, most people can even manage the difference between a stick shift and an automatic,

Because they fundamentally understand their common super class, the transmission.

People interface with encapsulated features on cars all the time. The brake and gas pedals hide an incredible array of complexity with an interface so simple you can operate them with your feet! The implementation of the engine, the style of brakes, and the size of the tires have no effect on how you interface with the class definition of the pedals. The final attribute, polymorphism, is clearly reflected in the ability of car manufacturers to offer a wide array of options on basically the same vehicle. For example, you can get an antilock braking system or traditional brakes, power or rack-and-pinion steering, 4-, or 6-, or 8-cylender engines. Either way, you will still press the break pedal to stop, turn the steering wheel to change direction, and press the accelerator when you want to move.

PROJECT TESTING

COMPILING TEST

It was a good idea to do our stress testing early on, because it gave us time to fix some of the unexpected deadlocks and stability problems that only occurred when components were exposed to very high transaction volumes.

EXECUTION TEST

This program was successfully loaded and executed. Because of good programming there were no execution errors. The complete performance of the project "INTRANET CHATTING" was good.

OUTPUT TEST

The successful output screens are placed in the output screens section above with brief explanation about each screen.

FUTURE IMPROVEMENT

1. This project can be enhanced by implementing different protocols and can be made more useful for varied clients according to the requirements of the client, it can also possible in future that each client in this globe has his own customized "INTRANET CHATTING".

2. It can be enhanced in the field of voice chatting. Using VoIP protocol
3. It can be enhanced in the field of Video Conferencing.

CONCLUSION

Even though this application has been developed with the users own Protocols, this can be used in an Intranet based organization.

1. This system was developed so that people can exchange information as well as converse with each other.
2. Through this system people can access chat rooms globally.
3. The system is interactive and friendly.
4. Entire system is fully automatic to the clients and satisfies the clients request
5. Especially the system is more useful to the technical people when the need for sending pictures, images it is solved through WHITE BOARD UTILITY OF "INTRANET CHATTING".