

Remote Desktop Control



Acknowledgement

First and foremost, we would like to thank to our supervisor of this project, Mr. Rajat Budhiraja for the valuable guidance and advice. He inspired us greatly to work in this project. His willingness to motivate us contributed tremendously to our project.

Besides, we would like to thank the staff members of INSTITUTE OF INFORMATICS & COMMUNICATION for providing us with a good environment and facilities to complete this project. It gave us an opportunity to participate and learn about the working of REMOTE DESKTOP CONTROL. The guidance and support received from all the members who contributed and who are contributing to this project, was vital for the surge success of the project.

Contents

- Introduction
- Where is it used?
- Network Management Categories Followed
- Flowchart
- Project Explanation
 - Server Side
 - Client Side
- Screenshots
- Source Code
 - Server Side
 - Client Side
- Libraries and APIs

Introduction

Remote Desktop Control Application is a client/server software package allowing remote network access to graphical desktop. This application enables you to get a view of the remote machine desktop and thus control it with your local mouse and keyboard. It can be used to perform remote system control and administration tasks in Unix, Windows and other assorted network environments.

This application requires a TCP/IP connection between the server and the viewer (client), which works on LANs. Each computer has a unique IP address and may also have a name in the DNS. User will need to know the IP address or name of the server when a viewer wants to connect to it. The initial handshaking consists of Client Initialization and Server Initialization messages. When the connection between a client and a server is first established, the server begins by requesting authentication from the client using a challenge-response scheme, which typically results in the user being prompted for a password at the client end.

The server is designed to make the client as simple as possible, so it is usually up to the server to perform any necessary translations. Each desktop is like a virtual X display, with a root window on which several X applications can be displayed. Servers mirror the real display to a remote client, which means that the server is not 'multi-user'. It does, however, provide the primary user of a PC with remote access to their desktop. The server processing includes retrieving the Desktop Image and sending it.

The input side is based on a standard workstation model of a keyboard, and multi-button pointing device. Input events are sent to the server by the client whenever the user presses a key or whenever the mouse is moved. It also requests of all the possible specific parameters that a server can handle, for instance the color mode, pointer events and so on.

WHERE IS IT USED?

- ✓ It can also be used for "headless computers". Instead of each computer having its own monitor, keyboard and mouse, a monitor, keyboard and mouse can be attached to one computer with remote control software, and headless computers controlled by it. The duplicate desktop mode is useful for user support and education.
- ✓ Remote control software combined with telephone communication can be nearly as helpful for novice computer-users as if the support staff were actually there.
- ✓ This is widely used by many computer manufacturers and large businesses' help desks for technical troubleshooting of their customers' problems.
- ✓ For an enterprise, Remote Desktop Services allow IT departments to install applications on a central server rather than on individual desktops. Remote users can log on and use them via network. This makes upgrading, troubleshooting and software management much easier.
- ✓ With Remote Desktop solution, problems within an organization can be addressed almost instantly notwithstanding the administrator's current spot. This business software also eradicates the need for large teams of IT administration at each location. Thus remote connection management is much more efficient.
- ✓ Remote Desktop capacitates a business owner to check the programs and files on his work computer while having a grand time at his vacation house. It offers flexibility that aids in getting better results while slicing operational costs.
- ✓ Remote access saves time. Instead of getting someone to email the file that you desperately need to finish your work, you can just access your computer and copy the file onto your work computer.

Network Management Categories Followed

Out of the five major functional areas of Network Management, the project focuses on Configuration and Security Management

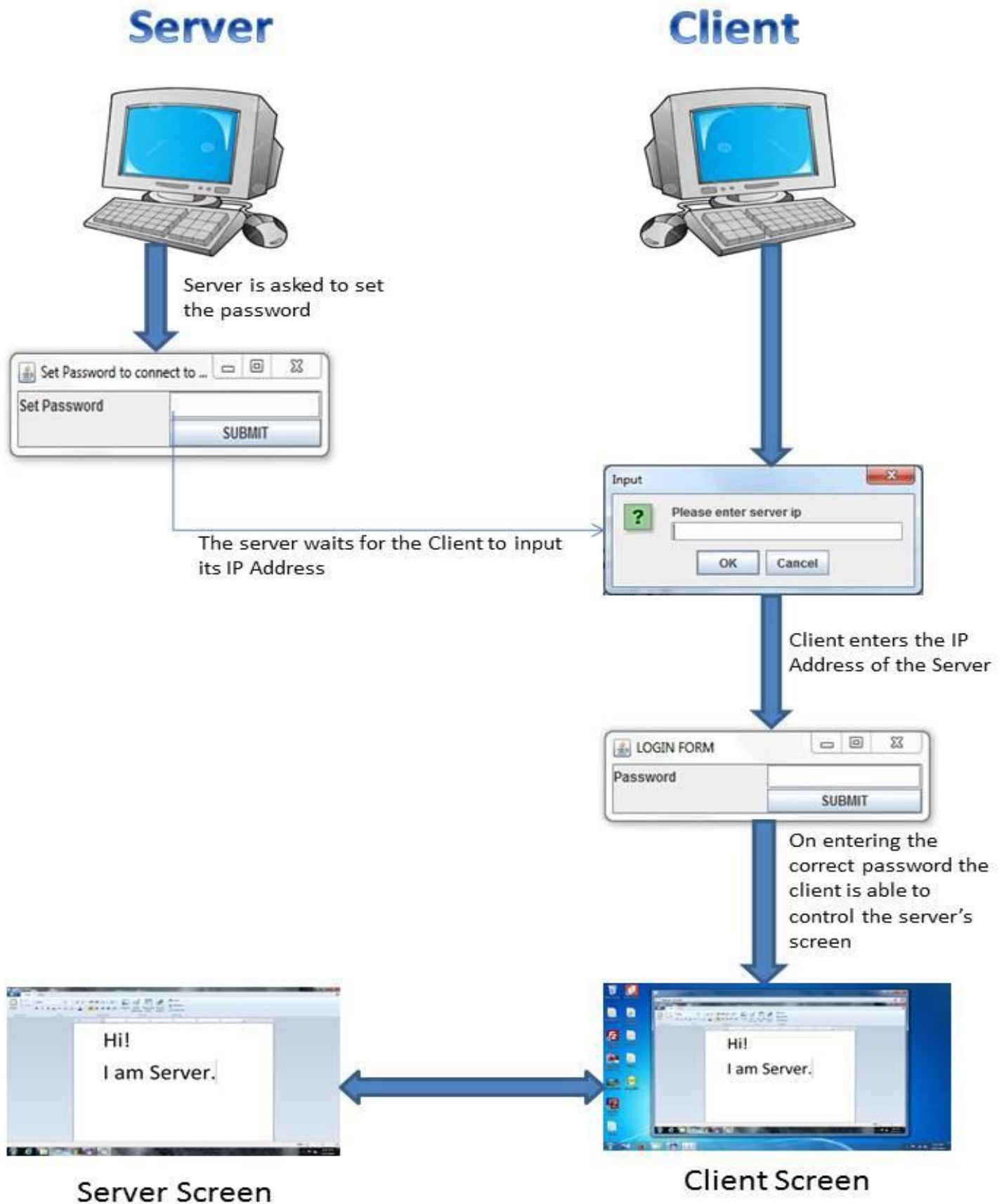
Configuration Management:

This application provides you with the feature to configure any System as Server/Client. The Port is initialized to a default value over which communication takes place.

Security Management:

In this application the user at the server end sets the password which is to be input by the user at the client end, thus, enabling a secured connection.

Flowchart



Project Explanation

SERVER SIDE

Start Class

This is the entry class which determines the GUI seen at the server side i.e. the size and location of the frame. This start class instantiates SetPassword class.

SetPassword Class

This class sets the password at the server side to connect with the client. Once the user at the server side enters the password and clicks the submit button. Then the password is set and the user at the client side must enter the same password in order to remotely control the server's desktop. This SetPassword class instantiates InitConnection class.

InitConnection Class

This class listens to server port and waits for clients connections. This class instantiates a robot device with dimensions of the screen size which are further passed on to the **SendScreen** class for capturing the screen.

These dimension of the screen are also passed on to the client from this class so that the client can make use of these dimensions and can make the mouse movements to be precisely at the same point where we want them to be (in case of different screen sizes). This class then instantiates two threads of SendScreen and ReceiveEvents.

SendScreen Class

SendScreen class continuously captures the screenshots of the server's desktop and sends them to the client, therefore runs as a separate thread.

The SendScreen has to continuously capture the server's desktop screenshots and write it to the server socket's output stream. So we use a loop here, which will loop infinitely till the server is shut down. The write method in ImageIO class is static and it writes the image by encoding it in the specific encoding (jpeg here).

ReceiveEvents Class

This class is responsible for receiving the keyboard and mouse events send by the client and simulating them in the server p.c. Since this class has to continuously receive the events therefore it has to be a thread.

The client sends a numeric variable (e.g. -1 for key press, -2 for mouse movement etc) with each event as to differentiate b/w different events and after recognizing various (using switch and enum) events from one another this class simulates them with the help of robot class methods as shown in the code snippet.

Commands class

Define constants which are used to represents client's commands.

CLIENT SIDE

Class Start

The start class has the main method. Thus the start class creates the client instance. Once started the server ip address is asked. This IP is of the system you want to access remotely.

When the connection with the server is established, the control shifts to the authenticate Class by instantiating its object.

Class Authenticate

This class ensures the security of the Application User. The Client authenticates itself by writing the Password which is verified at the Server side. Thus a user can access the Server only if they know the correct Authentication parameters. The Password can be set only at the server Side, giving all authority to the server.

If the Client inputs are matched, the CreateFrame Class is instantiated.

If the Client inputs do not match, the Connection is Lost and the Client will have to restart the application.

Class CreateFrame

This Class is responsible for creating the GUI for each connected Server. The major GUI components include JdesktopPane, JFrame & Internal JFrame. This class is a thread and binds an input stream to the socket connected.

The GUI Components and InputStream are passed as parameters to the Class ReceiveScreen, which receives the screenshots of server screen and displays them.

The CreateFrame also instantiates SendEvents class, thereby sending the mouse and keyboard events to the server for Simulating them on the server desktop.

Class ReceiveScreen

This class receives the screenshots from the server through the inputstream and then displays it. ReceiveScreen is again a separate thread. It follows the following steps to receive and display the screenshots.

- Connect to the server.
- Get InputStream from server
- Read bytes received from server till the end of image is encountered. JPEG images end with bytes -1 or -39.
- Decode the image in memory.
- Display the image in a JPanel inside the interframe.

Since the Client has to receive screenshots from the server continuously, we have a loop in this class that will poll infinitely till the server is shutdown.

Initially it receives all the data related to an image in a byte buffer and when the end of image is reached it stops filling the buffer and converts the byte buffer data into an image using ImageIO.read method with byteArrayInputStream.

Class SendEvents

This Class is responsible for sending all Client Actions(Mouse & Keyboard Actions) to the server so that they can be implemented on the server side using robot class methods. All mouse and keyboard events with their distinguished codes(Enum Commands) are captured using various Listeners.(MouseListener, KeyListener, MousemotionListener).

The Mouse movements are scaled before sending them to the server. Scaling is necessary to simulate the mousemovements accurately. Because of difference in screen dimensions of the ClientGUI over which events are recorded and Server desktop where they are simulated, appropriate scaling is important .

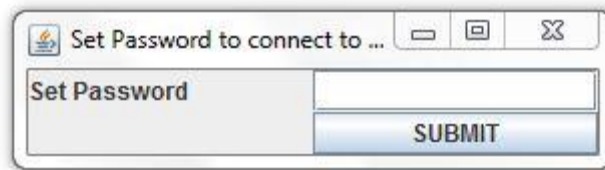
Here Scaling is done by multiplying the captured mouse movements with the actual Server desktop dimension which are received at the beginning(start class) and dividing those movements by the Client GUI i.e. JPanel dimensions.

Class Commands

Defines constants to distinguish and represent the various mouse and keyboard actions.


Screenshots

Server Side

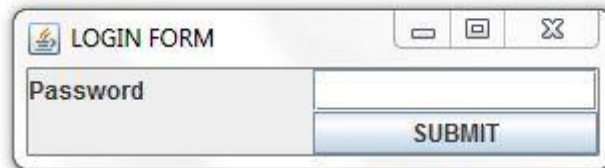


A dialog box titled "Set Password to connect to ...". It contains a label "Set Password" and a text input field. Below the input field is a blue button labeled "SUBMIT".

Client Side



A dialog box titled "Input". It contains a green question mark icon and the text "Please enter server ip". Below this is a text input field. At the bottom are two buttons: "OK" and "Cancel".



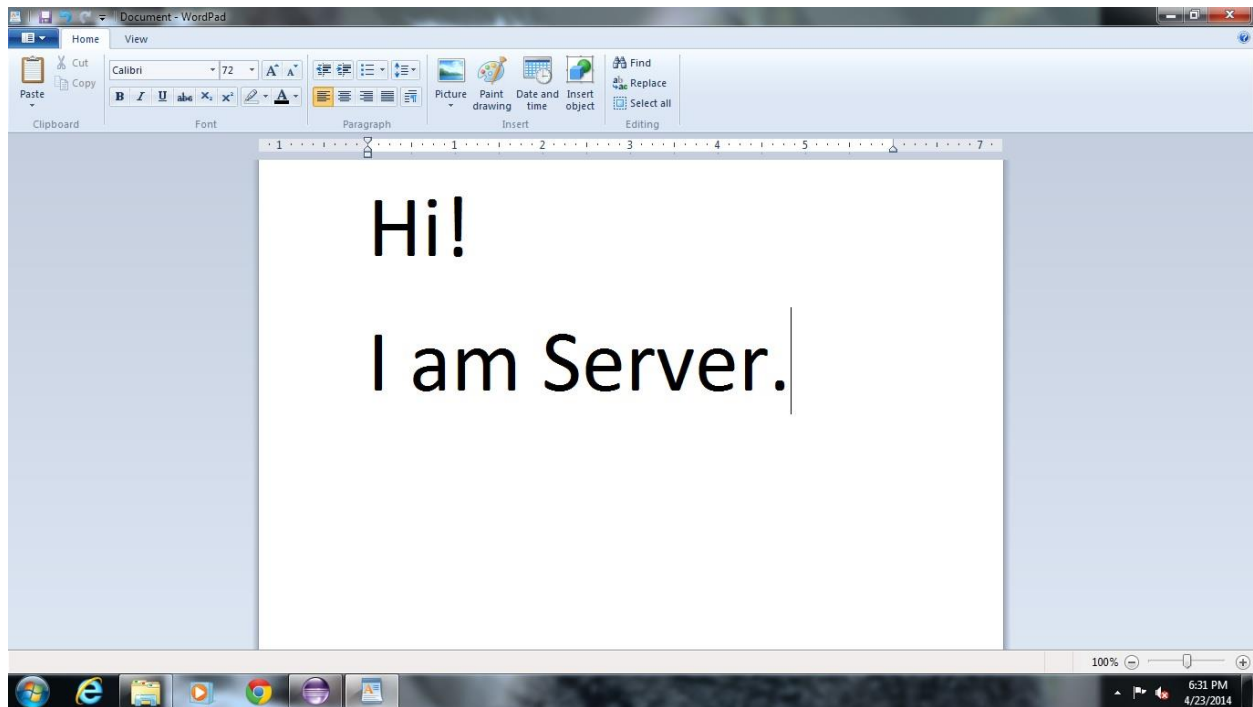
A dialog box titled "LOGIN FORM". It contains a label "Password" and a text input field. Below the input field is a blue button labeled "SUBMIT".

Incorrect Password entry

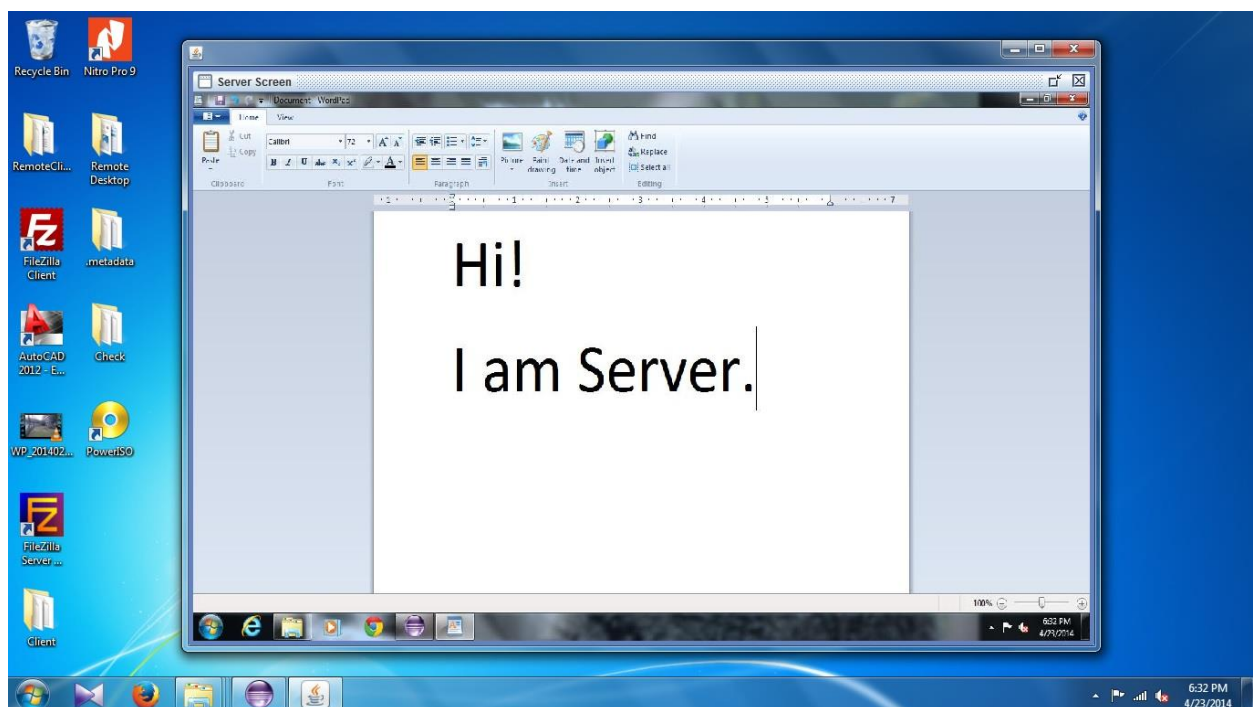


An error dialog box titled "Error". It contains a red octagonal icon with a white "X" and the text "Incorrect password". Below this is a blue button labeled "OK".

Server Screen



Client Screen



Code

SERVER SIDE

Start.Java

```
public class start{  
    public static void main(String[] args){  
        SetPassword frame1= new SetPassword();  
        frame1.setSize(300,80);  
        frame1.setLocation(500,300);  
        frame1.setVisible(true);  
    }  
}
```

SetPassword.Java

```
import javax.swing.*.*;  
import java.awt.event.*;  
import java.awt.*.*;  
  
public class SetPassword extends JFrame implements ActionListener{  
    static String port="4907";  
    JButton SUBMIT;  
    JPanel panel;  
    JLabel label1,label2;  
    JTextField text1,text2;  
    String value1, value2;  
    JLabel label;  
    SetPassword(){  
        label1=new JLabel();  
        label1.setText("Set Password");  
        text1 = new JTextField(15);
```

```

        label=new JLabel();
        label.setText("");
        this.setLayout(new BorderLayout());
        SUBMIT = new JButton("SUBMIT");
        panel=new JPanel(new GridLayout(2,1));
        panel.add(label1);
        panel.add(text1);
        panel.add(label);
        panel.add(SUBMIT);
        add(panel,BorderLayout.CENTER);
        SUBMIT.addActionListener(this);
        setTitle("Set Password to connect to the Client");
    }

    public void actionPerformed(ActionEvent ae){
        value1=text1.getText();
        dispose();
        new InitConnection(Integer.parseInt(port),value1);
    }

    public String getValue1(){
        return value1;
    }

    public static void main(String[] args){
        SetPassword frame1= new SetPassword();
        frame1.setSize(300,80);
        frame1.setLocation(500,300);
        frame1.setVisible(true);
    } }

```

InitConnection.java

```
import java.awt.Dimension;

import java.awt.GraphicsDevice;

import java.awt.GraphicsEnvironment;

import java.awt.GraphicsDevice;

import java.awt.Rectangle;

import java.awt.Robot;

import java.awt.Toolkit;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.net.ServerSocket;

import java.net.Socket;

import javax.swing.*.*;

public class InitConnection{

    ServerSocket socket = null;

    DataInputStream password = null;

    DataOutputStream verify = null;

    String width="";

    String height="";

    InitConnection(int port,String value1){

        Robot robot = null;

        Rectangle rectangle = null;

        try{

            System.out.println("Awaiting Connection from Client");

            socket=new ServerSocket(port);

            GraphicsEnvironment gEnv =

GraphicsEnvironment.getLocalGraphicsEnvironment();

            GraphicsDevice gDev = gEnv.getDefaultScreenDevice();
```



```

Dimension dim=Toolkit.getDefaultToolkit().getScreenSize();
String width="" + dim.getWidth();
String height="" + dim.getHeight();
rectangle=new Rectangle(dim);
robot=new Robot(gDev);
drawGUI();
while(true){
    Socket sc=socket.accept();
    password=new DataInputStream(sc.getInputStream());
    verify=new DataOutputStream(sc.getOutputStream());
    //String username=password.readUTF();
    String pssword=password.readUTF();
    if(pssword.equals(value1)){
        verify.writeUTF("valid");
        verify.writeUTF(width);
        verify.writeUTF(height);
        new SendScreen(sc,robot,rectangle);
        new ReceiveEvents(sc,robot);}
    else{
        verify.writeUTF("Invalid");
    }
}
} catch (Exception ex){
    ex.printStackTrace();
}
}

private void drawGUI(){
}
}

```

SendScreen.Java

```
import java.awt.Rectangle;
import java.awt.Robot;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.OutputStream;
import java.net.Socket;
import javax.imageio.ImageIO;

class SendScreen extends Thread{
    Socket socket=null;
    Robot robot=null;
    Rectangle rectangle=null;
    boolean continueLoop=true;
    OutputStream oos=null;
    public SendScreen(Socket socket,Robot robot,Rectangle rect) {
        this.socket=socket;
        this.robot=robot;
        rectangle=rect;
        start();
    }
    public void run(){
        try{    oos=socket.getOutputStream();
        }catch(IOException ex){
            ex.printStackTrace();
        }

        while(continueLoop){
            BufferedImage image=robot.createScreenCapture(rectangle);
```

```

try{
    ImageIO.write(image,"jpeg",oos);
}catch(IOException ex){
    ex.printStackTrace();
}

try{
    Thread.sleep(10);
}catch(InterruptedException e){
    e.printStackTrace();
}

}

}

}

```

ReceiveEvents.Java

```

import java.awt.Robot;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;

/* Used to receive server commands then execute them at the client side*/

class ReceiveEvents extends Thread{
    Socket socket= null;
    Robot robot = null;
    boolean continueLoop = true;
    public ReceiveEvents(Socket socket, Robot robot){
        this.socket = socket;
        this.robot = robot;
        start(); //Start the thread and hence calling run method
    }
}

```

```

}

public void run(){
    Scanner scanner = null;

    try {
        scanner = new Scanner(socket.getInputStream());
        while(continueLoop){
            //recieve commands and respond accordingly
            int command = scanner.nextInt();
            switch(command){
                case-1:
                    robot.mousePress(scanner.nextInt());
                    break;
                case-2:
                    robot.mouseRelease(scanner.nextInt());
                    break;
                case-3:
                    robot.keyPress(scanner.nextInt());
                    break;
                case-4:
                    robot.keyRelease(scanner.nextInt());
                    break;
                case-5:
                    robot.mouseMove(scanner.nextInt(),scanner.nextInt());
                    break;
            }
        }
    }catch(IOException ex){

```

```
        ex.printStackTrace();
    }
} //end function
} //end class
```

Commands.Java

```
public enum Commands{
    PRESS_MOUSE(-1),
    RELEASE_MOUSE(-2),
    PRESS_KEY(-3),
    RELEASE_KEY(-4),
    MOVE_MOUSE(-5);
    private int abbrev;
    Commands(int abbrev){
        this.abbrev = abbrev;
    }
    public int getAbbrev(){
        return abbrev;
    }
}
```

CLIENT SIDE

Start.Java

```
import java.net.Socket;

import javax.swing.JOptionPane;

public class Start{

    static String port = "4907";

    public static void main(String args[]){

        String ip = JOptionPane.showInputDialog("Please enter server ip");

        new Start().initialize(ip, Integer.parseInt(port));

    }

    public void initialize(String ip, int port){

        try{

            Socket sc = new Socket(ip,port);

            System.out.println("Connecting to the Server");

            //Authenticate class is responsible for security purposes

            Authenticate frame1= new Authenticate(sc);

            frame1.setSize(300,80);

            frame1.setLocation(500,300);

            frame1.setVisible(true);

        } catch (Exception ex){

            ex.printStackTrace();

        }

    }

}
```

Authenticate.Java

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

class Authenticate extends JFrame implements ActionListener{

    private Socket cSocket = null;

    DataOutputStream psswrchk = null;

    DataInputStream verification = null;

    String verify = "";

    JButton SUBMIT;

    JPanel panel;

    JLabel label, label1;

    String width="",height="";

    final JTextField text1;

    Authenticate(Socket cSocket){

        label1=new JLabel();

        label1.setText("Password");

        text1 = new JTextField(15);

        this.cSocket = cSocket;

        label=new JLabel();

        label.setText("");

        this.setLayout(new BorderLayout());

        SUBMIT = new JButton("SUBMIT");
```

```
panel=new JPanel(new GridLayout(2,1));  
panel.add(label1);  
panel.add(text1);  
panel.add(label);  
panel.add(SUBMIT);  
add(panel,BorderLayout.CENTER);  
SUBMIT.addActionListener(this);  
setTitle("LOGIN FORM");  
}
```

```
public void actionPerformed(ActionEvent ae){  
    String value1=text1.getText();  
    try{  
        psswrchk= new DataOutputStream(cSocket.getOutputStream());  
        verification= new DataInputStream(cSocket.getInputStream());  
        psswrchk.writeUTF(value1);  
        verify=verification.readUTF();  
    }catch (IOException e){  
        e.printStackTrace();  
    }  
    if(verify.equals("valid")){  
        try{  
            width = verification.readUTF();  
            height = verification.readUTF();  
        }catch (IOException e){  
            e.printStackTrace();  
        }  
        CreateFrame abc= new CreateFrame(cSocket,width,height);
```



```

        dispose();
    }
    else {
        System.out.println("enter the valid password");
        JOptionPane.showMessageDialog(this, "Incorrect password", "Error",
JOptionPane.ERROR_MESSAGE);
        dispose();
    }
}
}
}

```

CreateFrame.Java

```

import java.awt.BorderLayout;
import java.beans.PropertyVetoException;
import java.io.InputStream;
import java.net.Socket;
import javax.swing.JDesktopPane;
import javax.swing.JFrame;
import javax.swing.JInternalFrame;
import javax.swing.JPanel;
import java.util.zip.*;
import java.io.IOException;

class CreateFrame extends Thread {
    String width="", height="";
    private JFrame frame = new JFrame();

    //JDesktopPane represents the main container that will contain all connected clients' screens

    private JDesktopPane desktop = new JDesktopPane();

```

```

private Socket cSocket = null;

private JInternalFrame interFrame = new JInternalFrame("Server Screen", true, true, true);

private JPanel cPanel = new JPanel();

public CreateFrame(Socket cSocket, String width, String height) {

    this.width = width;

    this.height = height;

    this.cSocket = cSocket;

    start();

}

//Draw GUI per each connected client

public void drawGUI() {

    frame.add(desktop, BorderLayout.CENTER);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Show thr frame in maximized state

    frame.setExtendedState(frame.getExtendedState() | JFrame.MAXIMIZED_BOTH);
//CHECK THIS LINE

    frame.setVisible(true);

    interFrame.setLayout(new BorderLayout());

    interFrame.getContentPane().add(cPanel, BorderLayout.CENTER);

    interFrame.setSize(100,100);

    desktop.add(interFrame);

    try {

        //Initially show the internal frame maximized

        interFrame.setMaximum(true);

    }catch (PropertyVetoException ex) {

        ex.printStackTrace();

    }

}

//This allows to handle KeyListener events

```

```

        cPanel.setFocusable(true);

        interFrame.setVisible(true);
    }

    public void run() {

        //Used to read screenshots

        InputStream in = null;

        //start drawing GUI

        drawGUI();

        try{

            in = cSocket.getInputStream();

        }catch (IOException ex){

            ex.printStackTrace();

        }

        //Start receiving screenshots

        new ReceiveScreen(in,cPanel);

        //Start sending events to the client

        new SendEvents(cSocket,cPanel,width,height);

    }

}

```

ReceiveScreen.Java

```

import java.awt.Graphics;

import java.awt.Image;

import java.io.ByteArrayInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.ObjectInputStream;

import javax.imageio.ImageIO;

```

```

import javax.swing.JPanel;

class ReceiveScreen extends Thread{

    private ObjectInputStream cObjectInputStream = null;

    private JPanel cPanel = null;

    private boolean continueLoop = true;

    InputStream oin = null;

    Image image1 = null;

    public ReceiveScreen(InputStream in,JPanel p){

        oin = in;

        cPanel = p;

        start();

    }

    public void run(){

        try{

            //Read screenshots of the client and then draw them

            while(continueLoop){

                byte[] bytes = new byte[1024*1024];

                int count = 0;

                do{

                    count+=oin.read(bytes,count,bytes.length-count);

                }while(!(count>4 && bytes[count-2]==(byte)-1 && bytes[count-1]==(byte)-

39));

                image1 = ImageIO.read(new ByteArrayInputStream(bytes));

                image1 =

image1.getScaledInstance(cPanel.getWidth(),cPanel.getHeight(),Image.SCALE_FAST);

                //Draw the received screenshots

                Graphics graphics = cPanel.getGraphics();

                graphics.drawImage(image1, 0, 0, cPanel.getWidth(), cPanel.getHeight(),

cPanel); }

```

```
        } catch(IOException ex) {  
            ex.printStackTrace();  
        }  
    } }  
}
```

SendEvents.Java

```
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
import java.awt.event.MouseEvent;  
import java.awt.event.MouseListener;  
import java.awt.event.MouseMotionListener;  
import java.io.IOException;  
import java.io.PrintWriter;  
import java.net.Socket;  
import javax.swing.JPanel;  
  
class SendEvents implements KeyListener, MouseMotionListener, MouseListener{  
    private Socket cSocket = null;  
    private JPanel cPanel = null;  
    private PrintWriter writer = null;  
    String width = "", height = "";  
    double w,h;  
    SendEvents(Socket s, JPanel p, String width, String height){  
        cSocket = s;  
        cPanel = p;  
        this.width = width;  
        this.height = height;  
    }  
}
```

```

w = Double.valueOf(width.trim()).doubleValue();

h = Double.valueOf(width.trim()).doubleValue();

//Associate event listeners to the panel

cPanel.addKeyListener(this);

cPanel.addMouseListener(this);

cPanel.addMouseMotionListener(this);

try{

    //Prepare PrintWriter which will be used to send commands to the client

    writer = new PrintWriter(cSocket.getOutputStream());

    } catch(IOException ex) {

        ex.printStackTrace();

    }

}

public void mouseDragged(MouseEvent e){

}

public void mouseMoved(MouseEvent e){

    double xScale = (double)w/cPanel.getWidth();

    double yScale = (double)h/cPanel.getHeight();

    writer.println(Commands.MOVE_MOUSE.getAbbrev());

    writer.println((int)(e.getX()*xScale));

    writer.println((int)(e.getY()*yScale));

    writer.flush();

}

public void mouseClicked(MouseEvent e){

}

```

```
public void mousePressed(MouseEvent e){

    writer.println(Commands.PRESS_MOUSE.getAbbrev());

    int button = e.getButton();

    int xButton = 16;

    if(button==3){

        xButton = 4;

    }

    writer.println(xButton);

    writer.flush();

}

public void mouseReleased(MouseEvent e){

    writer.println(Commands.RELEASE_MOUSE.getAbbrev());

    int button = e.getButton();

    int xButton = 16;

    if(button==3){

        xButton = 4;

    }

    writer.println(xButton);

    writer.flush();

}

public void mouseEntered(MouseEvent e){

}

public void mouseExited(MouseEvent e){

}

public void keyTyped(KeyEvent e){
```

```

    }

    public void keyPressed(KeyEvent e){

        writer.println(Commands.PRESS_KEY.getAbbrev());

        writer.println(e.getKeyCode());

        writer.flush();

    }

    public void keyReleased(KeyEvent e){

        writer.println(Commands.RELEASE_KEY.getAbbrev());

        writer.println(e.getKeyCode());

        writer.flush();

    }

}

```

Commands.Java

```

public enum Commands{

    PRESS_MOUSE(-1),

    RELEASE_MOUSE(-2),

    PRESS_KEY(-3),

    RELEASE_KEY(-4),

    MOVE_MOUSE(-5);

    private int abbrev;

    Commands(int abbrev){

        this.abbrev = abbrev;

    }

    public int getAbbrev(){

        return abbrev;

    }

}

```


Libraries and API's

Java.net package

Provides the classes and interfaces for implementing networking applications. These include:

- TheURL class for basic access to Uniform Resource Locators (URLs).
- TheURLConnection class, which supports more complex operations on URLs.
- TheSocket class for connecting to particular ports on specific Internet hosts and reading and writing data using streams.
- TheServerSocket class for implementing servers that accept connections from clients.
- TheDatagramSocket, MulticastSocket, and DatagramPacket classes for implementing low-level networking.
- TheInetAddress class, which represents Internet addresses.
- Classes:

Class ServerSocket

java.net.ServerSocket

public class ServerSocket extends [Object](#) implements [Closeable](#)

This class implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

[ServerSocket](#)(int port)

Creates a server socket, bound to the specified port.

Class Socket

java.net.Socket

public class Socket extends [Object](#) implements [Closeable](#)

This class implements client sockets (also called just "sockets"). A socket is an endpoint for communication between two machines.

A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

getInputStream

public InputStream getInputStream() throws IOException

Returns an input stream for this socket.

If this socket has an associated channel then the resulting input stream delegates all of its operations to the channel. If the channel is in non-blocking mode then the input stream's read operations will throw an `IllegalBlockingModeException`.

Closing the returned `InputStream` will close the associated socket.

Returns: an input stream for reading bytes from this socket.

getOutputStream

public OutputStream getOutputStream() throws IOException

Returns an output stream for this socket.

If this socket has an associated channel then the resulting output stream delegates all of its operations to the channel. If the channel is in non-blocking mode then the output stream's write operations will throw an `IllegalBlockingModeException`.

Closing the returned `OutputStream` will close the associated socket.

Returns: an output stream for writing bytes from this socket.

PACKAGE: AWT

Abstract Window Toolkit (AWT) is a set of application program interfaces ([API](#) s) used by [Java](#) programmers to create graphical user interface ([GUI](#)) objects, such as buttons, scroll bars, and windows. AWT is part of the Java Foundation Classes ([JFC](#)) from Sun Microsystems, the company that originated Java. The JFC are a comprehensive set of GUI [class](#) libraries that make it easier to develop the user interface part of an application program.

Class Dimension

java.awt.Dimension

public class Dimension extends [Dimension2D](#) implements [Serializable](#)

The Dimension class encapsulates the width and height of a component (in integer precision) in a single object. The class is associated with certain properties of components. Several methods defined by the Component class and the LayoutManager interface return a Dimension object.

Class GraphicsDevice

java.awt. GraphicsDevice

public abstract class GraphicsDevice extends [Object](#)

The GraphicsDevice class describes the graphics devices that might be available in a particular graphics environment. These include screen and printer devices. Note that there can be many screens and many printers in an instance of [GraphicsEnvironment](#). Each graphics device has one or more [GraphicsConfiguration](#) objects associated with it. These objects specify the different configurations in which the GraphicsDevice can be used.

Class GraphicsEnvironment

java.awt. GraphicsEnvironment

public abstract class **GraphicsEnvironment** extends [Object](#)

The GraphicsEnvironment class describes the collection of [GraphicsDevice](#) objects and [Font](#) objects available to a Java(tm) application on a particular platform. The resources in this GraphicsEnvironment might be local or on a remote machine. GraphicsDevice objects can be screens, printers or image buffers and are the destination of [Graphics2D](#) drawing methods. Each GraphicsDevice has a number of [GraphicsConfiguration](#) objects associated with it. These objects specify the different configurations in which the GraphicsDevice can be used.

Class Image

java.awt. Image

public abstract class Image extends [Object](#)

The abstract class Image is the superclass of all classes that represent graphical images. The image must be obtained in a platform-specific manner.

Class BufferedImage

java.awt.image. BufferedImage

public class BufferedImage extends [Image](#) implements [WritableRenderedImage](#), [Transparency](#)

The BufferedImage subclass describes an [Image](#) with an accessible buffer of image data. A BufferedImage is comprised of a [ColorModel](#) and a [Raster](#) of image data. The

number and types of bands in the [SampleModel](#) of the Raster must match the number and types required by the ColorModel to represent its color and alpha components. All BufferedImage objects have an upper left corner coordinate of (0, 0). Any Raster used to construct a BufferedImage must therefore have minX=0 and minY=0.

Class Robot

java.awt.Robot

public class Robot extends [Object](#)

This class is used to generate native system input events for the purposes of test automation, self-running demos, and other applications where control of the mouse and keyboard is needed. The primary purpose of Robot is to facilitate automated testing of Java platform implementations.

Using the class to generate input events differs from posting events to the AWT event queue or AWT components in that the events are generated in the platform's native input queue. For example, Robot.mouseMove will actually move the mouse cursor instead of just generating mouse move events.

Constructor and Description

Robot()

Constructs a Robot object in the coordinate system of the primary screen.

Robot([GraphicsDevice](#) screen)

Creates a Robot for the given screen device.

Class Toolkit

public abstract class Toolkit extends [Object](#)

This class is the abstract superclass of all actual implementations of the Abstract Window Toolkit. Subclasses of the Toolkit class are used to bind the various components to particular native toolkit implementations.

Class Event

java.awt.event

public class EventObject extends Object implements Serializable

It is the root class from which all event state objects shall be derived. All Events are constructed with a reference to the object, the **source**, that is logically deemed to be the object upon which the Event in question initially occurred upon.

Class KeyEvent

java.awt.event.KeyEvent

public class KeyEvent extends InputEvent

An event which indicates that a keystroke occurred in a component. This low-level event is generated by a component object (such as a text field) when a key is pressed, released, or typed. The event is passed to every KeyListener or KeyAdapter object which registered to receive such events using the component's addKeyListener method. (KeyAdapter objects implement the KeyListener interface.) Each such listener object gets this KeyEvent when the event occurs.

"Key pressed" and "key released" events are lower-level and depend on the platform and keyboard layout. They are generated whenever a key is pressed or released, and are the only way to find out about keys that don't generate character input (e.g., action keys, modifier keys, etc.). The key being pressed or released is indicated by the getKeyCode and getExtendedKeyCode methods, which return a virtual key code.

Interface KeyListener

public interface KeyListener extends EventListener

The listener interface for receiving keyboard events (keystrokes). The class that is interested in processing a keyboard event either implements this interface (and all the methods it contains) or extends the abstract KeyAdapter class (overriding only the methods of interest).

The listener object created from that class is then registered with a component using the component's `addKeyListener` method. A keyboard event is generated when a key is pressed, released, or typed. The relevant method in the listener object is then invoked, and the `KeyEvent` is passed to it.

Class MouseEvent

`java.awt.event.MouseEvent`

`public class MouseEvent extends InputEvent`

`/** An event which indicates that a mouse action occurred in a component. This event is used both for mouse events (click, enter, exit) and mouse motion events (moves and drags).`

This low-level event is generated by a component object for:

- Mouse Events
 - a mouse button is pressed
 - a mouse button is released
 - a mouse button is clicked (pressed and released)
 - the mouse cursor enters a component
 - the mouse cursor exits a component
- Mouse Motion Events
 - the mouse is moved
 - the mouse is dragged

Interface MouseListener

`public interface MouseListener extends EventListener`

The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component. (To track mouse moves and mouse drags, use the `MouseMotionListener`.)

The class that is interested in processing a mouse event either implements this interface (and all the methods it contains) or extends the abstract `MouseAdapter` class (overriding only the methods of interest).

The listener object created from that class is then registered with a component using the component's `addMouseListener` method. A mouse

event is generated when the mouse is pressed, released clicked (pressed and released). A mouse event is also generated when the mouse cursor enters or leaves a component. When a mouse event occurs, the relevant method in the listener object is invoked, and the MouseEvent is passed to it.

Class ActionEvent

```
java.awt.event.ActionEvent  
public class ActionEvent extends AWTEvent
```

A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every every ActionListener object that registered to receive such events using the component's addActionListener method.

Interface ActionListener

```
public interface ActionListener extends EventListener
```

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

Class BorderLayout

```
java.awt.BorderLayout  
public class BorderLayout extends Object  
implements LayoutManager2, Serializable
```

A border layout lays out a container, arranging and resizing its components to fit in five regions: North, South, East, West, and Center. When adding a component to a container with a border layout, use one of these five names.

Class GridLayout

```
java.awt.GridLayout  
public class GridLayout extends Object  
implements LayoutManager, Serializable
```

The `GridLayout` class is a layout manager that lays out a container's Components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

JAVAX PACKAGES

`javax.imageio`

Class ImageIO

`javax.imageio.ImageIO`

public final class ImageIO extends Object

A class containing static convenience methods for locating ImageReaders and ImageWriters, and performing simple encoding and decoding.

SWING PACKAGES

`javax.swing`

Class JOptionPane

`javax.swing.JOptionPane`

public class JOptionPane extends JComponent implements Accessible

JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something.

Class JFrame

`javax.swing.JFrame`

public class JFrame extends Frame

implements WindowConstants, Accessible, RootPaneContainer

The class **JFrame** is an extended version of `java.awt.Frame` that adds support for the JFC/Swing component architecture.

The `JFrame` class is slightly incompatible with `Frame`. Like all other JFC/Swing top-level containers, a `JFrame` contains a `JRootPane` as its only child. The **content pane** provided by the root pane should, as a rule, contain all the non-menu components displayed by the `JFrame`. This is different from

the AWTFrame case.

Class JDesktopPane

javax.swing.JDesktopPane

public class JDesktopPane implements Accessible extends JLayeredPane

This class is normally used as the parent of JInternalFrames to provide a pluggable DesktopManager object to the JInternalFrames. The installUI of the L&F specific implementation is responsible for setting the desktopManager variable appropriately. When the parent of a JInternalFrame is a JDesktopPane, it should delegate most of its behavior to the desktopManager (closing, resizing, etc).

Class JInternalFrame

javax.swing.JInternalFrame

public class JInternalFrame

implements Accessible, WindowConstants, RootPaneContainer

extends JComponent

A lightweight object that provides many of the features of a native frame, including dragging, closing, becoming an icon, resizing, title display, and support for a menu bar.

Generally, you add JInternalFrames to a JDesktopPane. The UI delegates the look-and-feel-specific actions to the DesktopManager object maintained by the JDesktopPane.

Class JLabel

javax.swing.JLabel

public class JLabel

implements SwingConstants, Accessible extends JComponent

A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.

A JLabel object can display either text, an image, or both. You can specify

where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

Class JTextField

```
javax.swing.JTextField  
public class JTextField extends JTextComponent implements  
SwingConstants
```

The class **JTextField** is a component which allows the editing of a single line of text.

Class JPasswordField

```
javax.swing.JPasswordField  
public class JPasswordField  
extends JTextField
```

The JPasswordField class is a subclass of the JTextField class. As a more specialized version of the JTextField it behaves mainly the same as its parent class (i.e., a single line of editable text) except for masking the characters a user inputs. By default every character typed by the user is normally shown to be an asterisk (*).

I/O PACKAGE

Java.io package provides classes for system input and output through data streams, serialization and the file system.

Interface DataInput

```
java.io.DataInput  
public interface DataInput extends Object
```

DataInput is an interface describing streams that can read input in a machine-independent format.

Interface DataOutput

java.io.DataOutput

public interface DataOutput extends Object

DataOutput is an interface describing streams that can write output in a machine-independent format.

Class OutputStream

java.io.OutputStream

public abstract class OutputStream extends Object implements Closeable, Flushable

The **Java.io.OutputStream** class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink. Applications that need to define a subclass of OutputStream must always provide at least a method that writes one byte of output.

Class InputStream

java.io.InputStream

public abstract class InputStream extends Object implements Closeable

The **Java.io.InputStream** class is the superclass of all classes representing an input stream of bytes. Applications that need to define a subclass of InputStream must always provide a method that returns the next byte of input.

Class ByteArrayInputStream

java.io.ByteArrayInputStream

public class ByteArrayInputStream extends InputStream

The **java.io.ByteArrayInputStream** class contains an internal buffer that contains bytes that may be read from the stream. An internal counter keeps track of the next byte to be supplied by the read method. Following are the important points about ByteArrayInputStream:

Closing a ByteArrayInputStream has no effect.

The methods in this class can be called after the stream has been closed

without generating an IOException.

Class PrintWriter

java.io.PrintWriter

public class PrintWriter extends Writer

Prints formatted representations of objects to a text-output stream. This class implements all of the print methods found in PrintStream. It does not contain methods for writing raw bytes, for which a program should use unencoded byte streams.

Methods in this class never throw I/O exceptions, although some of its constructors may.

Class IOException

java.io.IOException

public class IOException extends Exception

Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

UTIL PACKAGES

Class Scanner

java.util.Scanner

public final class Scanner extends Object implements Iterator<String>

Scanner class is a simple text scanner which can parse primitive types and strings using regular expressions. Following are the important points about Scanner:

A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. A scanning operation may block waiting for input. A Scanner is not safe for multithreaded use without external synchronization.