**Documentación Del Proyecto Tipo Chat De POO.**

**Introducción:**

Este Programa realizado en java consta de 3 clases principales, donde se tocan los temas principales a tratar que son:

1. Uso del Thread (**Hilo**).
2. Manejo de archivos.

**Las 3 Clases Principales Son:**

1. La Clase **AppServer.**
2. La Clase **AppCliente.**
3. La Clase **RecibirArchivo.**

**Nota:** Las Clases y temas mencionados serán descritos más adelante en este trabajo.

**Alumno:** Anderson Gutiérrez

**C.I:** 29758990

**Catedra:** Programación Orientada a Objetos (**POO**)

**Parte N#1.** HILOS.

**Introducción (Para entrar en contexto).**

Un sistema operativo puede realizar multiprogramación al reasignar rápidamente tiempos de la CPU entre muchos programas, dando el aspecto de paralelismo, al ejecutarse concurrentemente. Aunque el verdadero paralelismo se logra con una computadora con varias CPU.

Java soporta varios hilos de ejecución. Un proceso de Java puede crear y manejar, dentro de sí mismo, varias secuencias de ejecución concurrentes o paralelos. Cada una de estas secuencias es un hilo independiente y todos ellos comparten tanto el espacio de dirección como los recursos del sistema operativo. Por lo tanto, cada hilo puede acceder a todos los datos y procedimientos del proceso, pero tiene su propio contador de programa y llamadas a métodos.

Todos los programas que hemos desarrollado hasta ahora (**Proyecto 1. Concesionario e Proyecto 2. Calculadora De Direcciones IP**) se ejecutan en forma secuencial, esto significa que, cuando llamamos a un método desde la main hasta que no finalice ese método no continúan ejecutándose las instrucciones de la main.

Debido a que para este proyecto (**Chat Cliente-Servido-Cliente**) necesitamos que el chat envié mensajes, imágenes, videos y maneje archivos, no sale rentable hacerlo de forma secuencial debido a su poca eficiencia de ejecución (**El proceso requiere mucho tiempo**). Para este proyecto necesitamos que el proceso de ejecución de los métodos y acciones sean concurrentes o paralelos para su correcto funcionamiento.

**¿Qué es un Thread (Hilo)?**

Un thread (**Hilo**) es un proceso que ha sido lanzado desde otro proceso y que ejecutará una secuencia de acciones concurrentemente o paralelamente a la ejecución del programa que lo lanzó (**Una forma de agilizar mucho los programas**).

**Como Crear Un Hilo En Java (Mi forma de crearlo).**

**Paso 1:** Debemos crear una clase que implemente la interface **'Runnable'**.

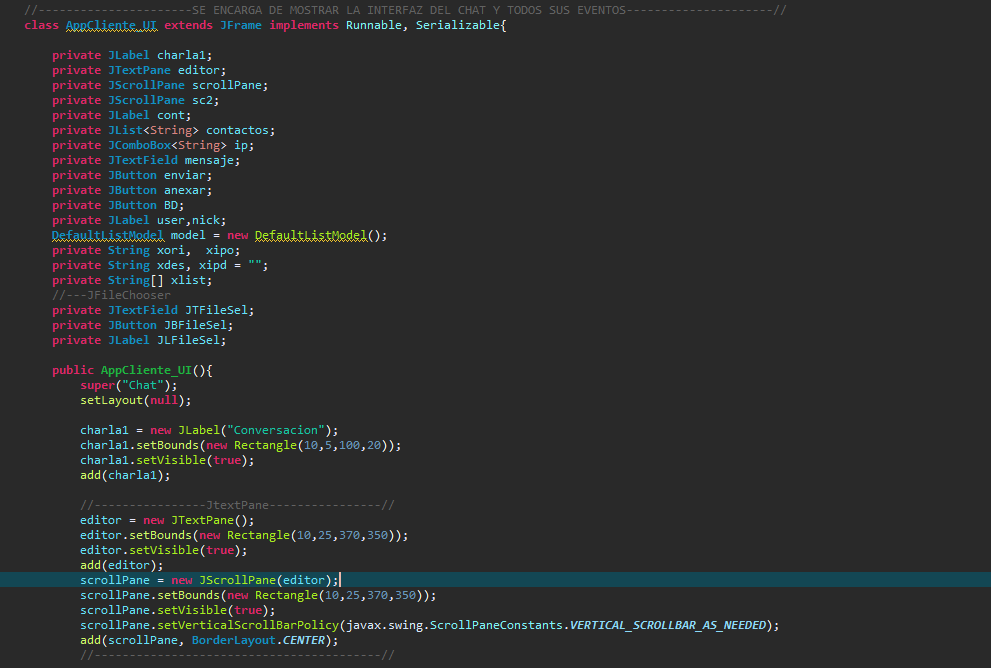
**Nota:** La interface **'Runnable'** nos obligara a implementar un método llamado **'run'**.

**Paso 2:** Dentro del constructor de la clase **'AppCliente\_UI'**, definimos un objeto de la clase Thread y al constructor de dicha clase le pasamos como referencia **'this'** que representa el objeto que implementa la interfaz ‘**Runnable’**.



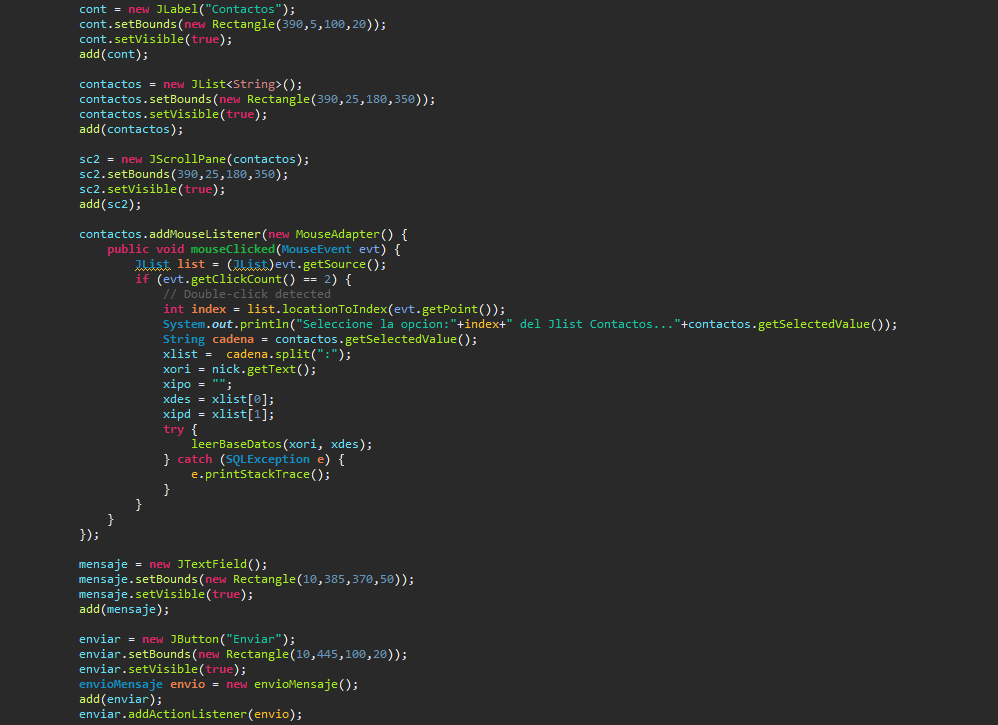
**C:\Users\Anderson\Desktop\Chat\hilo.PNGPaso 1**

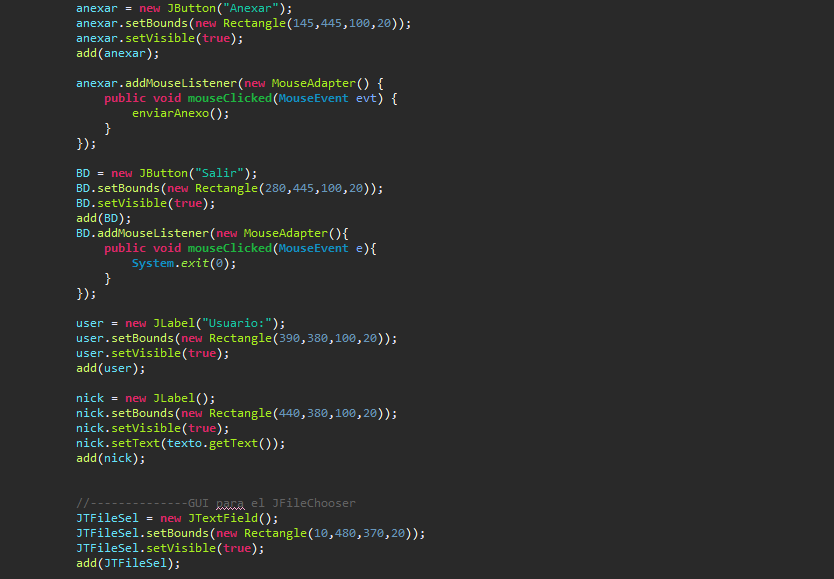
**Paso 2**

**Ejemplo Del Lado Del Cliente (Ejemplo Propio):**

**Paso 1**

**Dentro del constructor se crea el Thread.**







**Paso 2**

**Ejemplo Del Lado Del Server (Ejemplo Propia).**

**Paso 1**

**Dentro del constructor de crea el hilo**

**Paso 2**

**Parte N#2.** Manejo de Archivos.

**Introducción (Breve contextualización):**

Los programas que hemos hecho hasta ahora usan variables para almacenar información: los datos de entrada, los resultados calculados y valores intermedios generados a lo largo del cálculo. Toda esta información es efímera: cuando acaba el programa, todo desaparece. Pero, para muchas aplicaciones, es importante poder almacenar datos de manera permanente. Cuando se desea guardar información más allá del tiempo de ejecución de un programa lo habitual es organizar esa información en uno o varios ficheros almacenados en algún soporte de almacenamiento persistente. Otras posibilidades como el uso de bases de datos utilizan archivos como soporte para el almacenamiento de la información.

**Como Manejar Archivos en Java:**

**Para ello es importante diversas clases:**

Utilizamos esta clase para levantar la ventana del directorio y buscar el archivo.

1. **JFileChooser => Javax.swing. JFileChooser (Utilizada solo en el Cliente)**

Utilizada para seleccionar el archivo y crear una carpeta en el directorio, si no existe.

1. **File => Java.io.File (**Utiliza en el **Cliente** y en la clase **RecibirArchivo)**

Utilizadas para guardar la ruta (Path). (**Utilizadas solo en el** **Cliente**)

1. **Path => java.nio. file. Path**
2. **Paths => java.nio. file. Paths**

Utilizada para sobreescribir el fichero de destino si existe lo copia

1. **Files => java.nio. file. Files (Utilizadas solo en el Cliente)**

Utilizada para crear la dirección IP de la máquina que recibirá el archivo.

1. **InetAddress => java.net. InetAddres (Utilizadas solo en el Cliente)**

Utilizada para establecer comunicación con la clase RecibirArchivo. **(Utiliza en el Cliente y en la clase RecibirArchivo)**

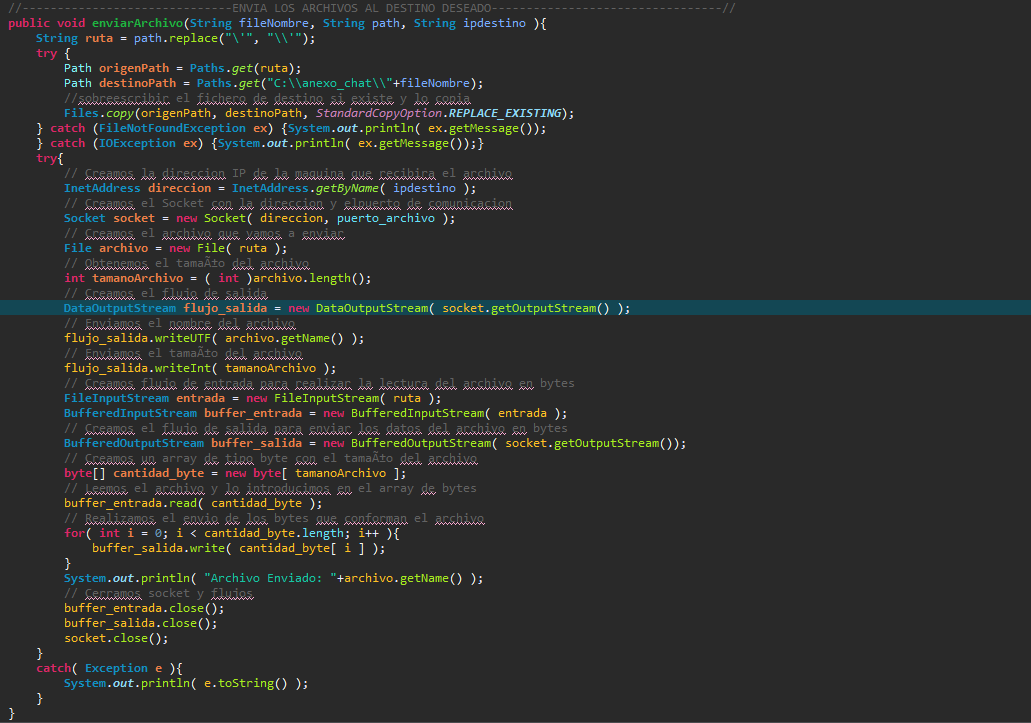
1. **Socket => java.net. Socket**
2. **ServerSocket => java.net. ServerSocket**

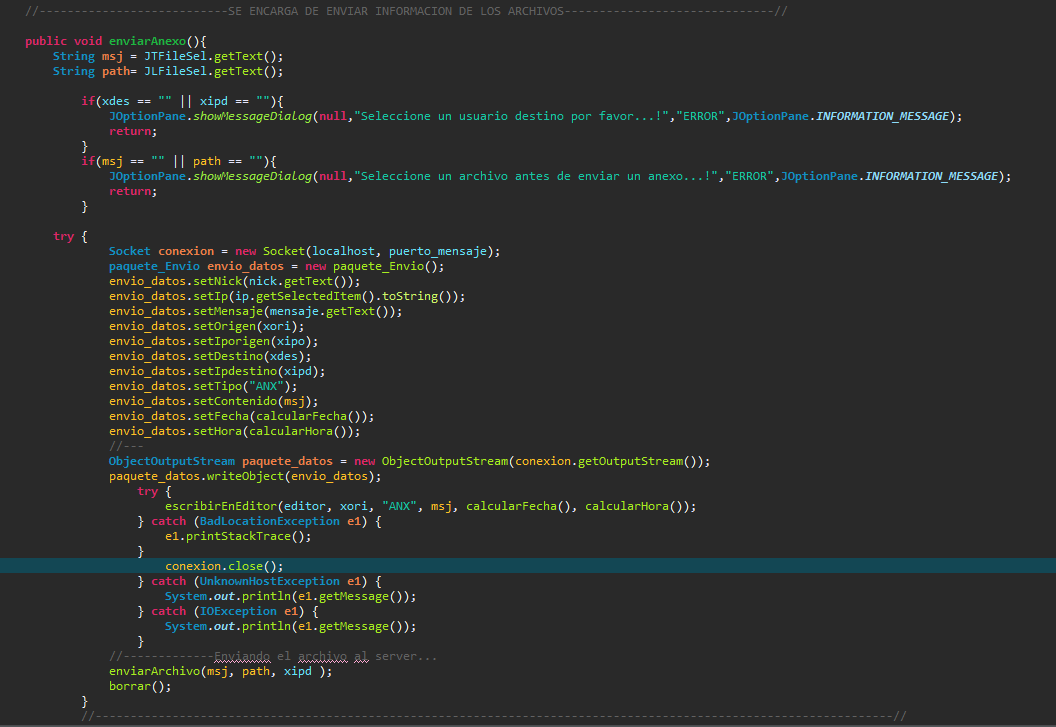
Utilizada para él envió de data. **(Utiliza en el Cliente y en la clase RecibirArchivo)**

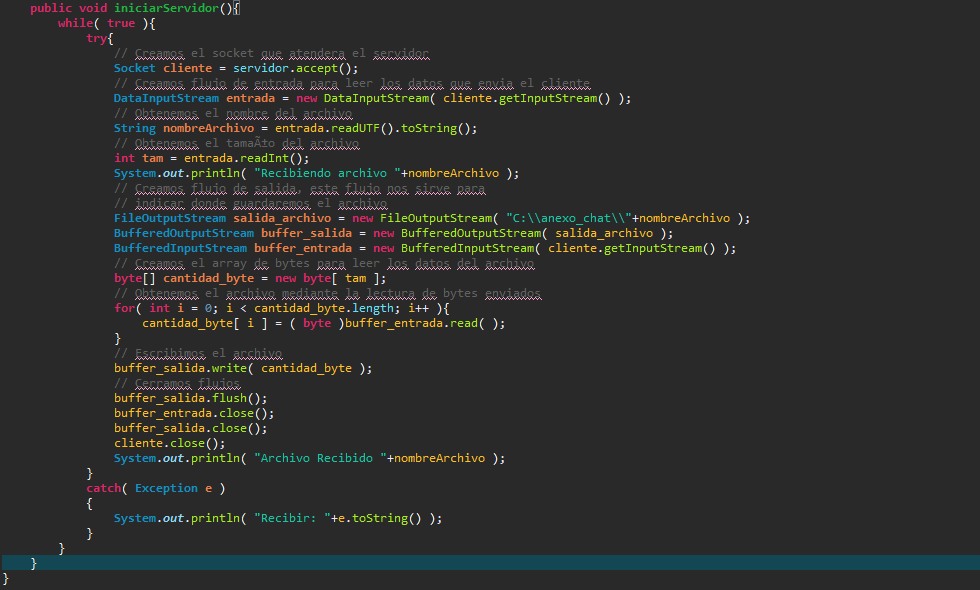
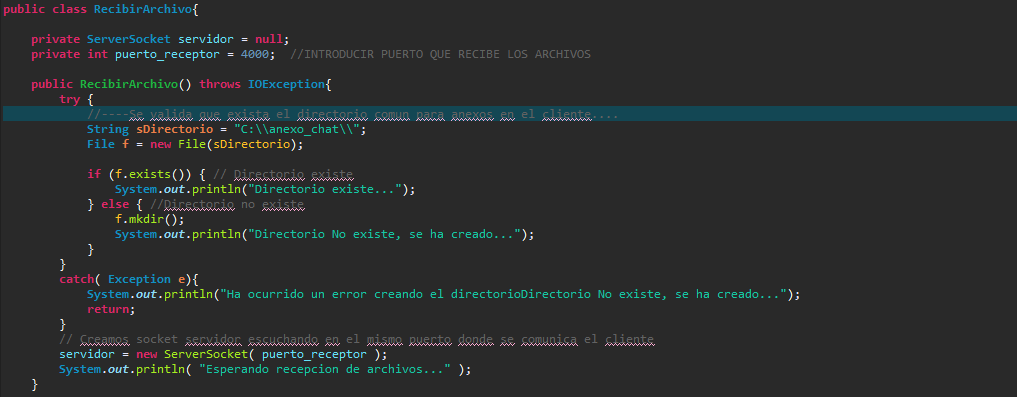
1. **DataOutputStream => java.io. DataOutputStream**
2. **BufferedOutputStream => java.io. BufferedOutputStream**

Utilizada para la recepción de data. **(Utiliza en el Cliente y en la clase RecibirArchivo)**

1. **FileInputStream => java.io. FileInputStream**
2. **BufferedInputStream => java.io. BufferedInputStream**
3. **DataInputStream => java.io. DataInputStream**

******Del Lado Del Cliente (CODIGO Propio):**

Método que se encargado de notificarle al servidor que se a envía un mensaje tipo Anexo **(Archivo).**

**Del Lado De La Clase Que Recibe Los Archivos (CODIGO Propio):**

**Parte N#3.** Envió y recepción De Mensajes.

**Descripción del servidor:**

Se establecen los tipos datos que se recibirán (**Variables**) de la clase **paquete\_Envio** del Cliente, luego se abre un bucle infinito (**while(true**)) para que se mantenga siempre a la escucha para ir recibiendo los datos que vengan llegando del cliente (**ObjectInputStream**), luego se verifica, si el mensaje que llego es “**IniciarSesion**”, se incluye el Nick del usuario y busca la dirección IP del usuario que se conectó (**InetAddress(getHostAddress)**), luego se crea un bucle (**Foreach**) para recorrer el **ArrayList** donde se insertaron el Nick del usuario que inicio sesión y su IP, luego se abre un **Socket** para enviar a algún cliente que ya este conectado el Nick y la IP de usuario que inicio sesión y se escribe en el servidor el mensaje recibido (**IniciarSesion**), el **Nick** y la **dirección IP.** Si el mensaje que llega no es **“IniciarSesion”** se busca la IP **(InetAddress(getHostAddress)),** se escribe en el servidor el **Mensaje, Nick y IP,** se insertan los datos en la base de datos y luego se abre un **Socket** para enviar los mensajes al cliente que ya esté conectado (**ObjectOutputStream**).

**Descripción del Cliente (PARTE EMISORA):**

Se crea una clase llamada **“EnvioSenal”,** que es la encargada de enviar el mensaje **InicarSesion** al registrarse en el chat, se crea otra clase llamada **“paquete\_Envio”** que es la contiene todos los datos que se enviaran al servidor (**GETTERS y SETTERS**). Por último se crea una clase llamada “**EnvioMensaje**” que la encargada de enviar a través de un **Socket** los mensajes que escriban los clientes y se escriben en el JTextPane del chat .

**NOTA**: El **JTextPane** permite colocar botones, colocar iconos, modificar letras, insertar líneas, etc.

**Descripción del Cliente (PARTE RECEPTORA):**

Se establecen los tipos datos que se recibirán **(Variables) del servidor, se abre un bucle infinito (while(true))** para que se mantenga siempre a la escucha para ir recibiendo los datos que vengan llegando del servidor **(ObjectInputStream),** si el mensaje que envía el servidor es **IniciarSesion** se coloca en el **JList** el **Nick** yla **IP** del usuario que se acaba de conectar**,** Si el mensaje que llega no es **"IniciarSesion"** es que ya están chateando y se colocan en el **JTextPane** en contenido de Las conversaciones.

**Clases Utilizadas**:

**Utilizada para establecer conexión.**

1. Socket => java.net. Socket
2. ServerSocket => java.net. ServerSocket

**Utilizada Para la recepción de Datos.**

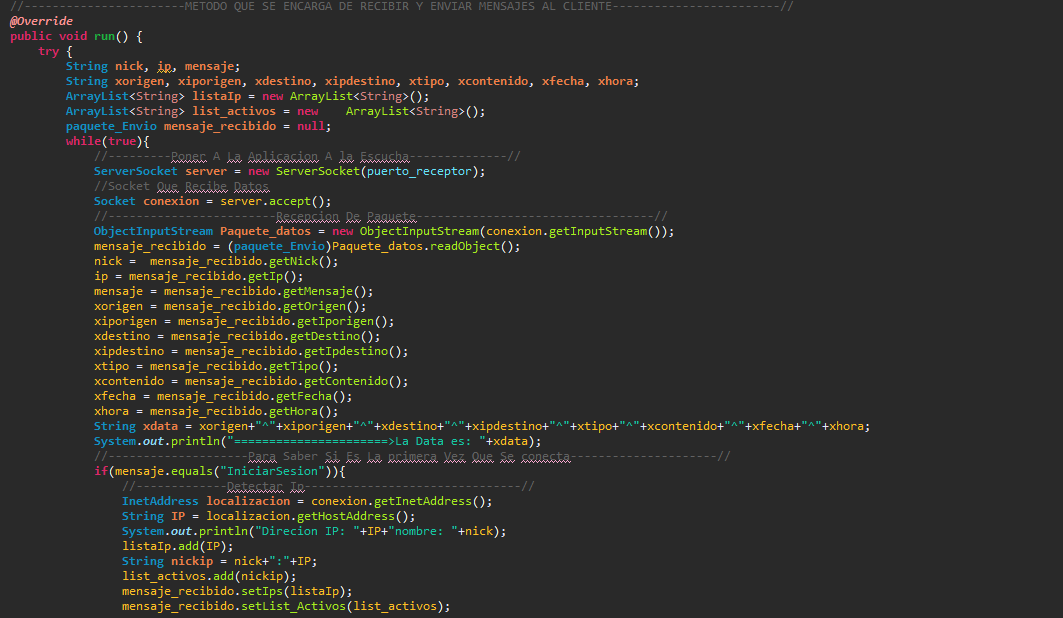
1. ObjectInputStream => java.io. ObjectInputStream

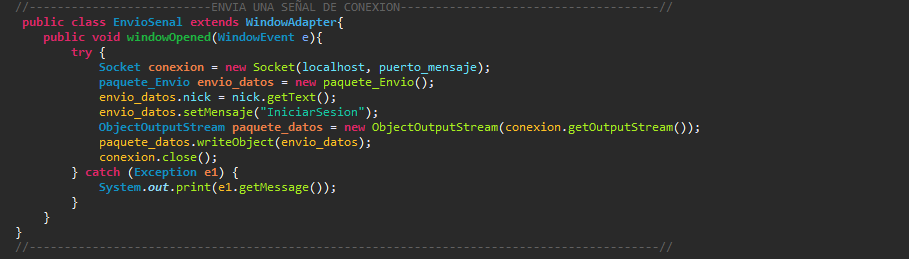
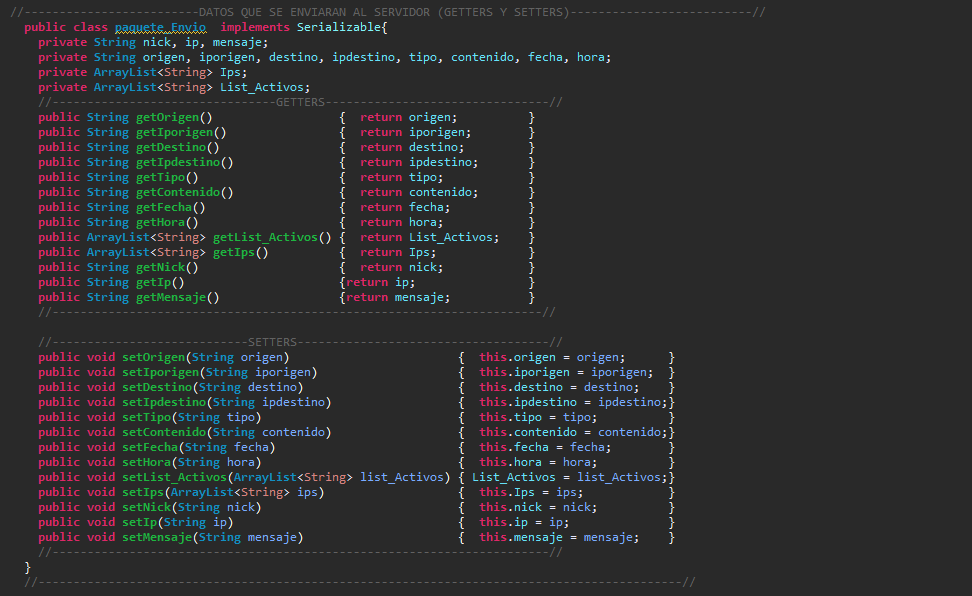
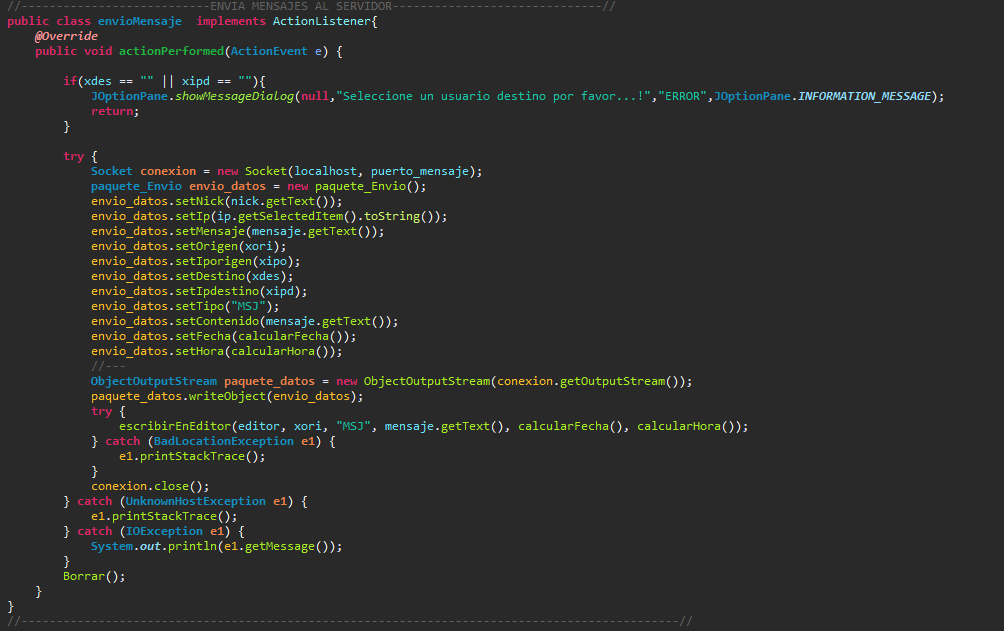
**Utilizada Para Enviar de Datos.**

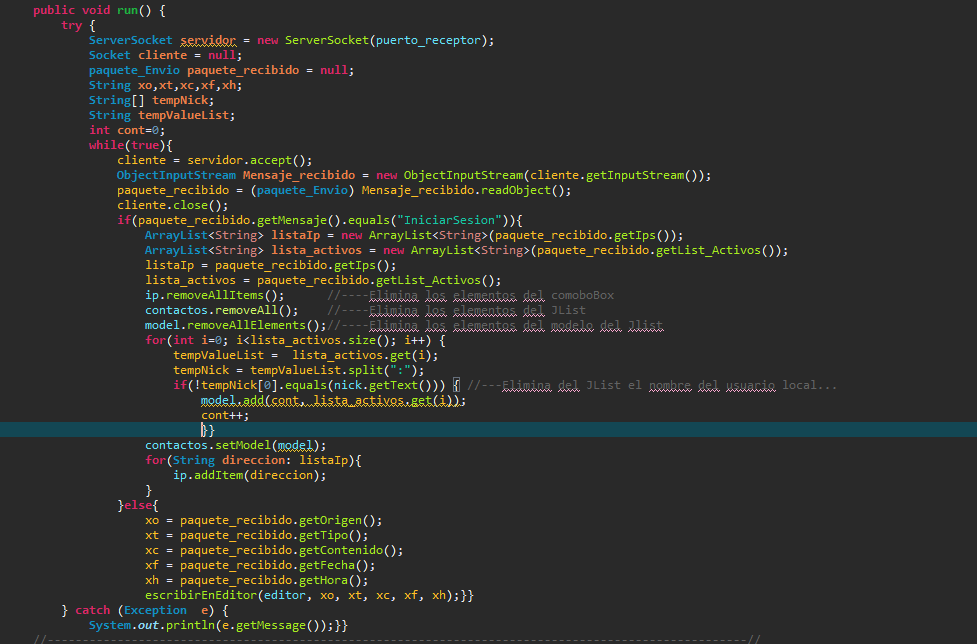
1. ObjectOutputStream => java.io. ObjectOutputStream

**Utilizada para crear la dirección IP de la máquina que recibirá el archivo.**

1. InetAddress => java.net. InetAddress (Utilizadas solo en el Cliente)

**Del Lado Del Servidor (CODIGO propio).**

**Del Lado Del Cliente (Parte EMISORA):**

**Del Lado Del Cliente (Parte RECEPTORA):** Método encargado de recibir los que envía el servidor.

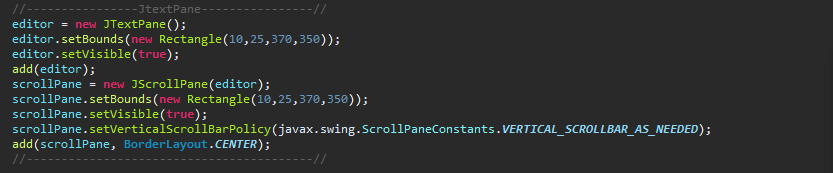
**Parte N#4**. Métodos Encargados De **Escribir, insertar líneas y abrir archivos** En El **JTextPane** Del Cliente.

**Descripción** **De** **Los** **Métodos**:

**La función del método** (**escribirEnEditor**): es el que verifica si el cliente que envía **el mensaje o el Anexo(Archivo)** es el **origen**, si el que envía es el origen se mostrara **la letra en negrita** en el **JTextPane**, si no, se muestra la ***letra en cursiva*** con el objetivo de poder distinguir de que quien proviene el mensaje o Archivo enviado.

**La función de los métodos** (**insertarNuevaLinea** y **insetarLineaDivisoria**): Son los métodos encargados se de colocar la línea que divide un mensaje de otro.

**La función del método (abrirArchivo):** es el método encargado de abrir el Anexo(**Archivo**) envido, al darle Click al botón **Ver Anexo** que se muestra en el **JTextPane.**

**JTextPane:** Admite texto con cierto formato, Admite texto plano, HTML y RTF por defecto, pero además permite que vayamos añadiendo texto cada uno con sus propios atributos de texto -fuentes, colores, etc-. Admite también directamente iconos de la clase Icon de java e incluso cualquier Component de java, como JButton, etc.

**Clases Utilizadas**:

**Utilizada para otorgar atributos a las frases.**

1. SimpleAttributeSet=>javax.swing.text. SimpleAttributeSet

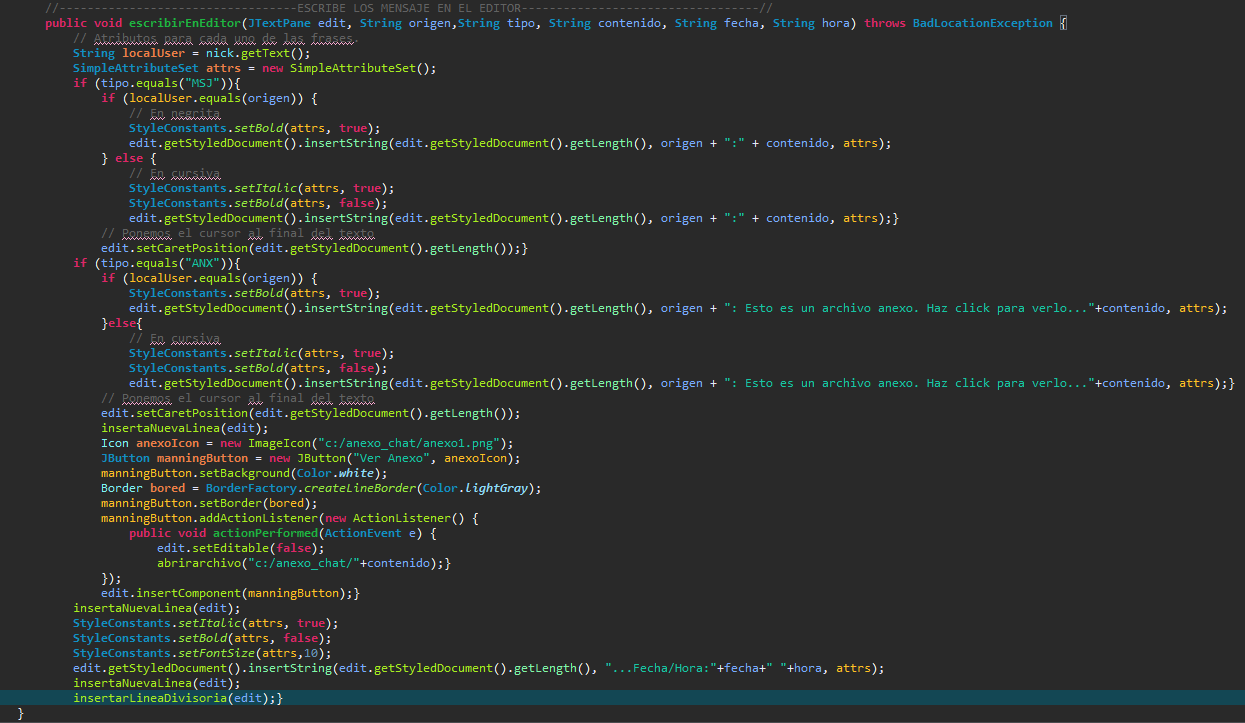
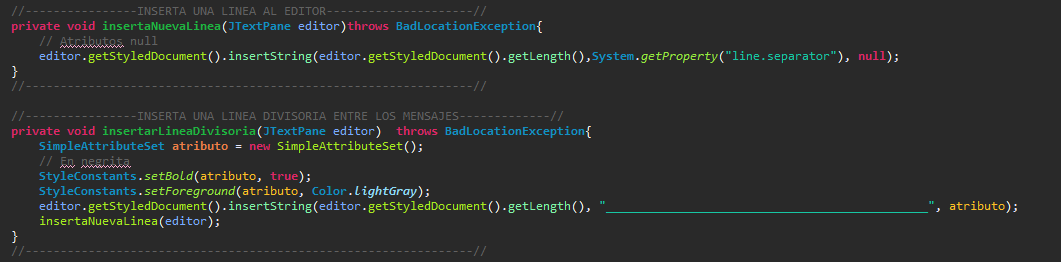
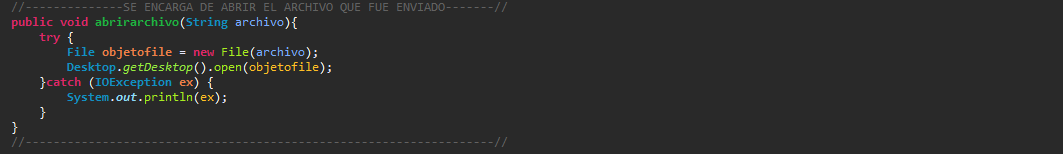
**Utilizada para modificar estilos De las letras**

1. StyleConstants => javax. swing. Text. StyleConstants

**Utilizada para abrir el anexo que es en el JTextPane.**

1. Desktop =>java.awt. Desktop

Métodos Encargados De **Escribir, insertar líneas y abrir archivos** En El **JTextPane** Del Cliente.

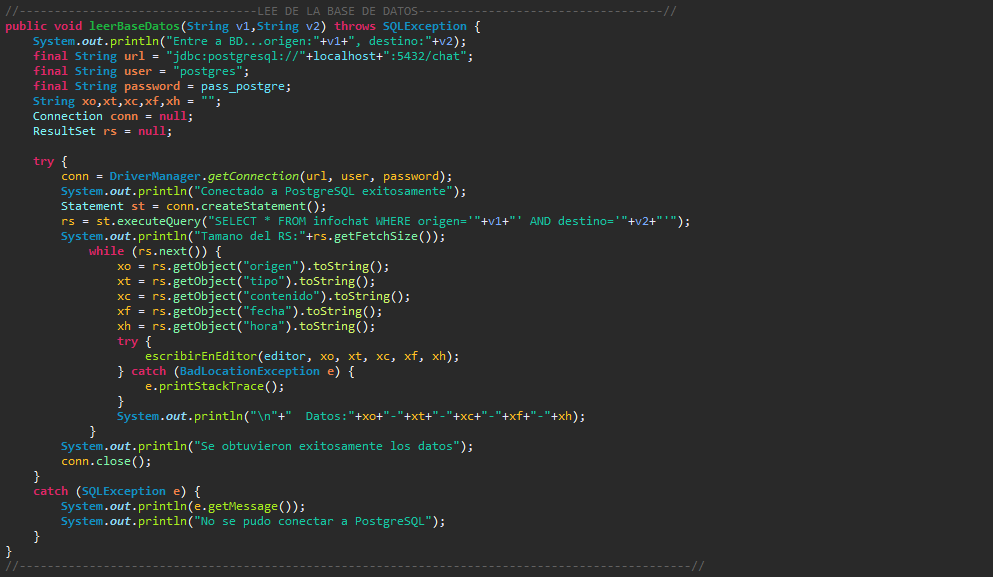


**Parte N#5.** Métodos encargados de Insertar y leer de base de datos.

**Clases Utilizadas:**

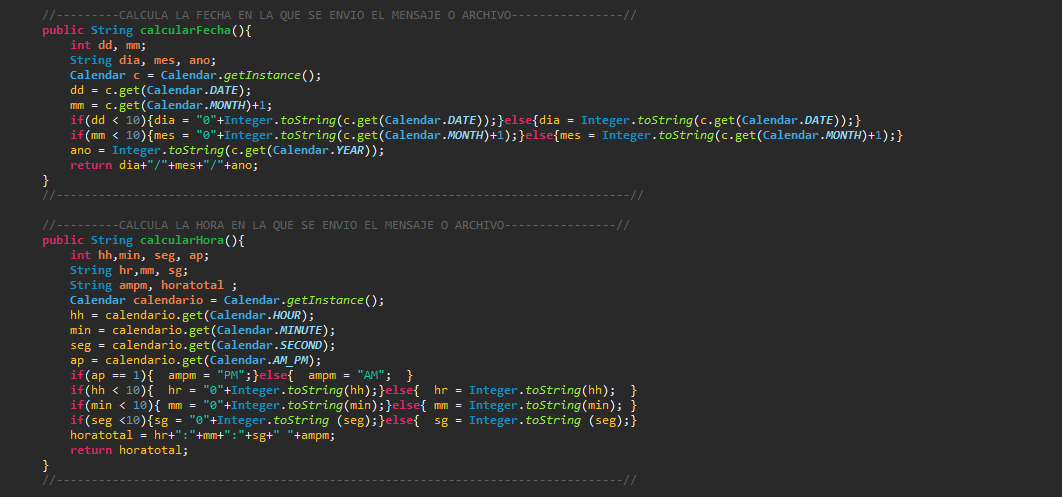
* Connection => java. sql. Connection
* ResultSet => java. sql. ResultSet
* Statement => java. sql. Statement

**Del Lado Del Servidor:**

**Del Lado Del Cliente:**

**Parte N#6.** Métodos encargados de calcular la fecha y hora en la que el mensaje fue enviado.

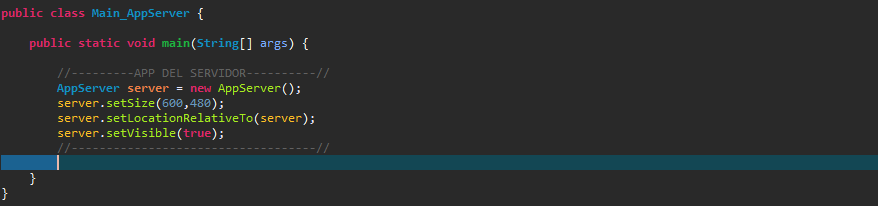
**Clase Utilizada:**

* ****Calendar => java. util. Calendar

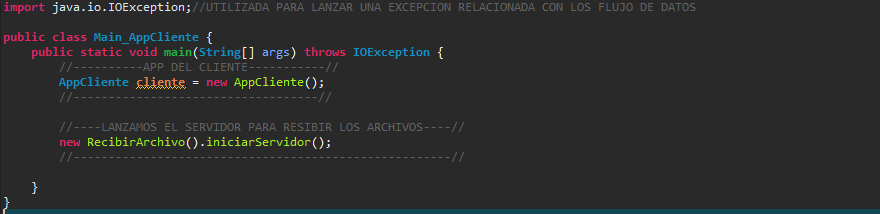
**Parte N#7.** Métodos encargados de borrar los **JTextField** seleccionados.

**Parte N#8**. Clases Encargadas de la ejecución del programa.

**Del Lado Del Servidor:**

**App:**

**Del Lado Del Cliente:**

**App:**