**Programación orientada a objetos III**

Ingeniería en Desarrollo de Software

Unidad: 3. Programación en red

Evidencia de aprendizaje. Programa cliente/servidor utilizando sockets

Francisco Manuel López García

Matrícula: ES1921010154

Profesor: Manuel Niño Navarrete

24 de noviembre 2021

Índice

[**Programa cliente/servidor utilizando sockets** 3](#_Toc88604587)

[**Introducción:** 3](#_Toc88604588)

[**Desarrollo:** 3](#_Toc88604589)

[*Investigación:* 3](#_Toc88604590)

[*Capturas de pantalla:* 7](#_Toc88604591)

[**Conclusión:** 20](#_Toc88604592)

[**Bibliografía:** 20](#_Toc88604593)

# **Programa cliente/servidor utilizando sockets**

# **Introducción:**

La evidencia de aprendizaje pretende crear un proyecto en NetBeans utilizando elementos básicos de la POO y el manejo de sockets para enviar datos entre dos aplicaciones cliente/servidor.

Para ello, y a partir de las indicaciones de la actividad se pone en práctica la manipulación de sockets a través del diseño y la programación de un proyecto informático en Java NetBeans que dé solución a la problemática planteada.

Este trabajo contiene el proyecto Java en Netbeans, capturas de pantalla de su ejecución y el código fuente.

Asimismo, se integra una conclusión del tema y se añaden las referencias bibliográficas consultadas.

# **Desarrollo:**

## *Investigación:*

La red informática es una red de conmutación de paquetes que tiene la ventaja de que se pueden transmitir varios paquetes a intercambiar en un cable, lo que reduce el costo de establecer una red y varias computadoras pueden compartir el mismo cable sin interferir entre sí. Además permite la subcontratación es que puede detectar si el paquete está dañado durante la transmisión.

El protocolo define un conjunto de reglas claras sobre cómo se comunican las computadoras (formato de dirección, cómo se empaquetan los datos, etc.) Por ejemplo, el Protocolo de transferencia de hipertexto (HTTP) define cómo los navegadores web se comunican con los servidores (programmerclick, 2019).

La programación en red tiene 3 elementos de comunicación

1. Dirección IP: se refiere al identificador único de la computadora en la red, 32 bits (4 bytes), que generalmente se expresa en decimal con puntos. La programación Java puede usar la clase InetAddress para manipular direcciones IP.
2. Número de puerto. Dirección lógica utilizada para identificar el proceso, la identidad de diferentes procesos. Puertos válidos: 0-65535, donde los sistemas 0 1024 usan o reservan puertos.
3. Protocolo de transmisión. Los protocolos de transmisión comunes son UDP (Protocolo de datagramas de usuario) y TCP (Protocolo de control de transmisión)

La mayor parte de la programación de red moderna se basa en un modelo cliente / servidor. Los datos se almacenan en un servidor o en una nube de servidor. La lógica de la interfaz es procesada por el software del cliente y el cliente se ejecuta en una computadora personal relativamente económica. El cliente inicia una conversación y el servidor espera a que el cliente le hable.

El estilo cliente/servidor establece una relación entre dos aplicaciones las cuales una de elle (cliente) inicia la comunicación y envía requerimientos a la otra aplicación (servidor) que los procesa y ejecuta y, si es necesario, envía la respuesta (De la Torre, et.al, 2010).

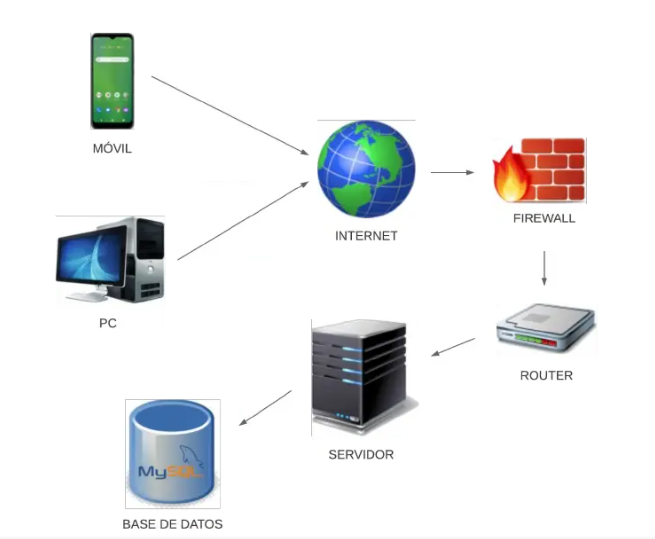
Sus características:

* Los componentes son clientes y servidores.
* Los servidores no conocen el número o identidades de los clientes sin embargo el cliente si conoce la identidad del servidor.
* Divide al sistema en tres entidades: una aplicación cliente, una aplicación servidor y una red de conexión.
* Los conectores están basados en protocolos de interacción entre redes. "Puede utilizar un amplio rango de protocolos y formatos de datos para comunicar la información" (De la Torre, et.al, 2010, p. 19).
* Los procesos que realiza el cliente es el envío de peticiones, procesamiento de la respuesta recibida, provee una interfaz gráfica para la comunicación directa con el usuario.
* El servidor no realiza ningún tipo de petición al cliente, sino que solamente envía datos de respuesta a las solicitudes realizadas, previa autentificación y verificación del cliente o usuario.

Programación con Sockets Api explícitamente utilizada, por aplicaciones del paradigma cliente/servidor. Hay dos tipos de servicio de transporte vía el API de socket:

* datagrama no fiable
* flujo de bytes fiable

Socket: una puerta entre procesos de aplicación y el protocolo de transporte extremo a extremo (UCP o TCP). Servicio TCP en Java, transferencia fiable de bytes desde un proceso a otro.



Los sockets stream ofrecen un servicio orientado a conexión, donde los datos se transfieren como un flujo continuo, sin encuadrarlos en registros o bloques. Este tipo de socket se basa en el protocolo TCP, que es un protocolo orientado a conexión. Esto implica que antes de transmitir información hay que establecer una conexión entre los dos sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita la conexión (cliente). Una vez que los dos sockets están conectados, ya se puede transmitir datos en ambas direcciones. El protocolo incorpora de forma transparente al programador la corrección de errores. Es decir, si detecta que parte de la información no llegó a su destino correctamente, esta volverá a ser trasmitida. Además, no limita el tamaño máximo de información a transmitir.

Establece un camino virtual entre servidor y cliente, fiable, sin perdidas de información ni duplicados, la información llega en el mismo orden que se envía.

* El cliente abre una sesión en el servidor y este guarda un estado del cliente.
* El cliente utiliza la clase Socket
* El servidor utiliza la clase ServerSocket
* Comunicación cliente/servidor con Socket TCP

El interfaz Java que da soporte a sockets TCP está constituida por las clases ServerSocket y Socket. ServerSocket: es utilizada por un servidor para crear un socket en el puerto en el que escucha las peticiones de conexión de los clientes. Su método accept toma una petición de conexión de la cola, o si la cola está vacía, se bloquea hasta que llega una petición. El resultado de ejecutar accept es una instancia de Socket, a través del cual el servidor tiene acceso a los datos enviados por el cliente. Socket: es utilizada tanto por el cliente como por el servidor. El cliente crea un socket especificando el nombre DNS del host y el puerto del servidor, así se crea el socket local y además se conecta con el servicio.

Esta clase proporciona los métodos getInputStream y getOutputStream para acceder a los dos streams asociados a un socket (recordemos que son bidireccionales), y devuelve tipos de datos InputStream y OutputStream, respectivamente, a partir de los cuales podemos construir Buffered Reader y PrintWriter, respectivamente, para poder procesar los datos de forma más sencilla.

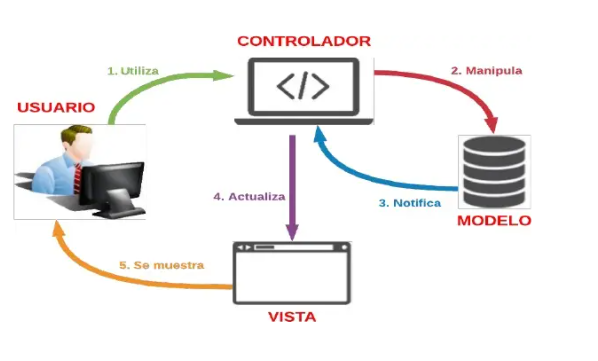
La interacción se da mediante la invocación / par de mensajes resultantes

***- Ejemplo***

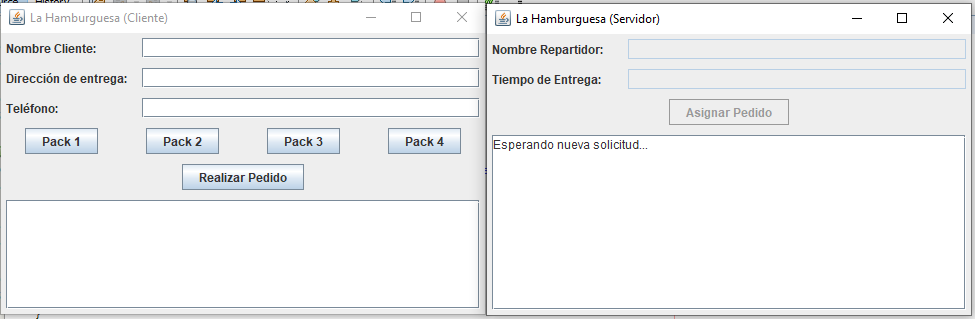
* Servidor http: cliente (navegador) página solicitada, servidor entrega página
  + Servicios de caching (servidores proxy)
* Caching de páginas Web frecuentemente utilizadas
  + Procesos pares (no cliente-servidor: peer-to-peer)
* Procesos que tienen en gran parte similitudes de funcionalidad

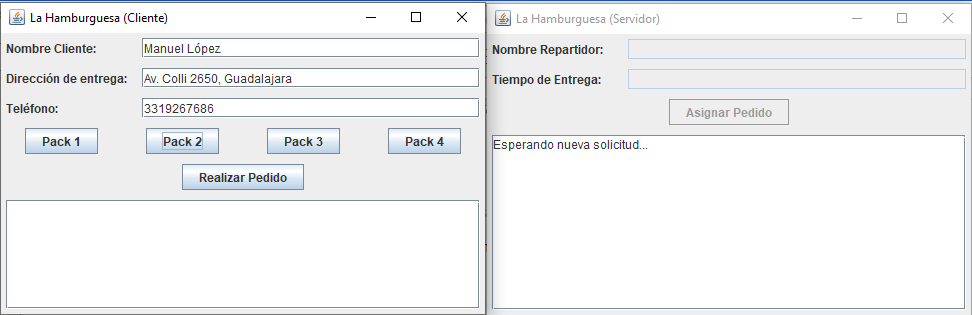
Las variantes del modelo cliente servidor

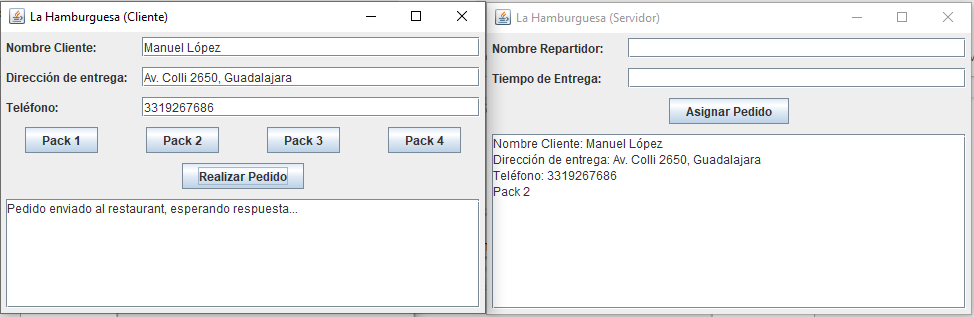
* Modelo MVC: creado para aplicaciones de escritorio y ampliamente adaptado para diseñar e implementar aplicaciones web.
* Servicios proporcionados por múltiples servidores (ej omercio web).
* Servidores proxy: suministrar replicación/distribución trasparente (caching. motores de búsqueda.
* Código Móvil código enviado a un proceso cliente para realizar una tarea especifica (E; Applets, Mensajes Activos

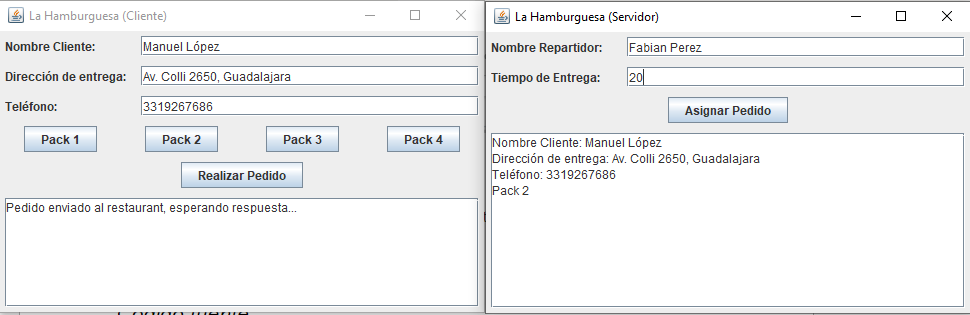
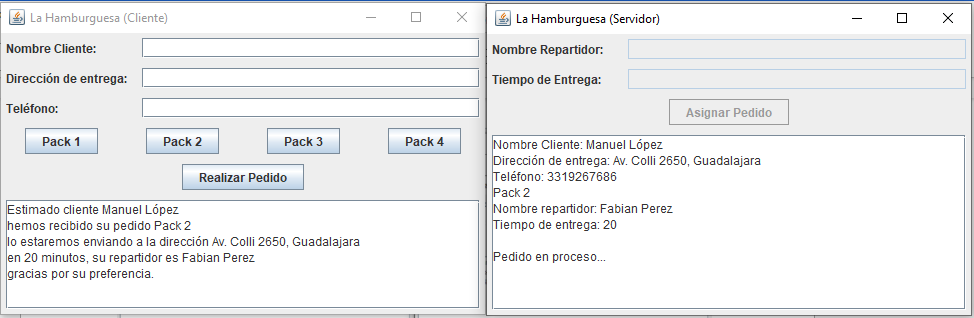


## *Capturas de pantalla:*

Inicio de la aplicación, se abre el cliente y el servidor se queda a la espera de la conexión  


Llena un pedido con cliente, dirección de entrega, teléfono y se selecciona un paquete  


Se pulsa realizar pedido, esto se envía al servidor que lo mostrara en su Caja de texto  


Se asigna repartidor y tiempo de entrega  
  
  
Se pulsa asignar pedido, con lo cual se envía la respuesta al cliente y el servidor se queda a la espera de una nueva solicitud  


*Código fuente:*

***Clase principal Main***

*package hamburguesa;*

*public class Hamburguesa {*

*public static void main(String[] args) {*

*java.awt.EventQueue.invokeLater(new Runnable() {*

*@Override*

*public void run() {*

*new HamburguesaServidor().setVisible(true);*

*}*

*});*

*new HamburguesaCliente().setVisible(true);*

*}*

*}*

***Clase HamburguesaCliente***

*package hamburguesa;*

*import java.io.DataInputStream;*

*import java.io.DataOutputStream;*

*import java.io.InputStream;*

*import java.io.OutputStream;*

*import java.net.Socket;*

*import javax.swing.JOptionPane;*

*public final class HamburguesaCliente extends javax.swing.JFrame {*

***// Declaración de variables para apariencia***

*javax.swing.JButton btnRealizarPedido;*

*javax.swing.JButton btnPack1;*

*javax.swing.JButton btnPack2;*

*javax.swing.JButton btnPack3;*

*javax.swing.JButton btnPack4;*

*javax.swing.JLabel lblNombreCliente;*

*javax.swing.JLabel lblDireccion;*

*javax.swing.JLabel lblTelefono;*

*javax.swing.JTextField txtNombreCliente;*

*javax.swing.JTextField txtDireccion;*

*javax.swing.JTextField txtTelefono;*

*javax.swing.JTextArea txtAreaEstatus;*

*javax.swing.JScrollPane ScrollEstatus;*

*int pack=0;*

***//Objeto que se comunicará con el servidor para enviarle datos.***

*Socket cliente;*

***//Objetos que permitirán leer y escribir flujos de bits través del socket.***

*InputStream entrada;*

*OutputStream salida;*

***//Objetos que permitirán leer valores de algún tipo desde el socket.***

*DataInputStream lector;*

*DataOutputStream escritor;*

***//Esta variable sirve para mantener o cerrar el ciclo de lectura de datos***

***//desde el cliente***

*boolean leyendo;*

*public HamburguesaCliente(){*

*initComponents();*

*abrirConexion();*

*}*

*private void initComponents() {*

***/\* En esta sección se tienen los elementos que compondrán la apariencia grafica***

***de la aplicación, declaraciones y acomodos en la pantalla \*/***

*java.awt.GridBagConstraints gridBagConstraints;*

*lblNombreCliente = new javax.swing.JLabel();*

*lblDireccion = new javax.swing.JLabel();*

*lblTelefono = new javax.swing.JLabel();*

*txtNombreCliente = new javax.swing.JTextField();*

*txtDireccion = new javax.swing.JTextField();*

*txtTelefono = new javax.swing.JTextField();*

*ScrollEstatus = new javax.swing.JScrollPane();*

*txtAreaEstatus = new javax.swing.JTextArea();*

*btnRealizarPedido = new javax.swing.JButton();*

*btnPack1 = new javax.swing.JButton();*

*btnPack2 = new javax.swing.JButton();*

*btnPack3 = new javax.swing.JButton();*

*btnPack4 = new javax.swing.JButton();*

*setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);*

*setTitle("La Hamburguesa (Cliente)");*

*lblNombreCliente.setText("Nombre Cliente:");*

*lblDireccion.setText("Dirección de entrega:");*

*lblTelefono.setText("Teléfono:");*

*txtAreaEstatus.setEditable(false);*

*btnPack1.setText("Pack 1");*

*btnPack1.addActionListener(new java.awt.event.ActionListener() {*

*@Override*

*public void actionPerformed(java.awt.event.ActionEvent evt) {*

*pack=1;*

*}*

*});*

*btnPack2.setText("Pack 2");*

*btnPack2.addActionListener(new java.awt.event.ActionListener() {*

*@Override*

*public void actionPerformed(java.awt.event.ActionEvent evt) {*

*pack=2;*

*}*

*});*

*btnPack3.setText("Pack 3");*

*btnPack3.addActionListener(new java.awt.event.ActionListener() {*

*@Override*

*public void actionPerformed(java.awt.event.ActionEvent evt) {*

*pack=2;*

*}*

*});*

*btnPack4.setText("Pack 4");*

*btnPack4.addActionListener(new java.awt.event.ActionListener() {*

*@Override*

*public void actionPerformed(java.awt.event.ActionEvent evt) {*

*pack=4;*

*}*

*});*

*btnRealizarPedido.setText("Realizar Pedido");*

*btnRealizarPedido.addActionListener(new java.awt.event.ActionListener() {*

*@Override*

*public void actionPerformed(java.awt.event.ActionEvent evt) {*

*realizarPedido();*

*}*

*});*

*getContentPane().setLayout(new java.awt.GridBagLayout());*

*setBounds(250, 160, 500, 320);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*gridBagConstraints.gridwidth = 2;*

*getContentPane().add(lblNombreCliente, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 0;*

*gridBagConstraints.gridy = 1;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*gridBagConstraints.gridwidth = 2;*

*getContentPane().add(lblDireccion, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 0;*

*gridBagConstraints.gridy = 2;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*gridBagConstraints.gridwidth = 2;*

*getContentPane().add(lblTelefono, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 1;*

*gridBagConstraints.gridy = 0;*

*gridBagConstraints.gridwidth = 3;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 20, 5, 5);*

*getContentPane().add(txtNombreCliente, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 1;*

*gridBagConstraints.gridy = 1;*

*gridBagConstraints.gridwidth = 3;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 20, 5, 5);*

*getContentPane().add(txtDireccion, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 1;*

*gridBagConstraints.gridy = 2;*

*gridBagConstraints.gridwidth = 3;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 20, 5, 5);*

*getContentPane().add(txtTelefono, gridBagConstraints);*

*txtAreaEstatus.setColumns(20);*

*txtAreaEstatus.setRows(5);*

*ScrollEstatus.setViewportView(txtAreaEstatus);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 0;*

*gridBagConstraints.gridy = 5;*

*gridBagConstraints.gridwidth = 4;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;*

*gridBagConstraints.weightx = 1.0;*

*gridBagConstraints.weighty = 1.0;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(ScrollEstatus, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 0;*

*gridBagConstraints.gridy = 3;*

*gridBagConstraints.weightx = 1.0;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(btnPack1, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 1;*

*gridBagConstraints.gridy = 3;*

*gridBagConstraints.weightx = 1.0;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(btnPack2, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 2;*

*gridBagConstraints.gridy = 3;*

*gridBagConstraints.weightx = 1.0;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(btnPack3, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 3;*

*gridBagConstraints.gridy = 3;*

*gridBagConstraints.weightx = 1.0;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(btnPack4, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 0;*

*gridBagConstraints.gridy = 4;*

*gridBagConstraints.gridwidth = 4;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(btnRealizarPedido, gridBagConstraints);*

*}*

***//Con esta función se mandan a llamar las funciones para enviar los datos***

*private void realizarPedido() {*

*if(pack != 0){*

*/****/Se llama la función para enviar datos***

*enviarDatos();*

***//Se llama la función para leer los datos del servidor***

*leerDatos();*

*}*

*else{*

*/****/Notificamos que no se ha seleccionado ningún pack***

*JOptionPane.showMessageDialog(this,"No se ha seleccionado el Pack a pedir","Error",JOptionPane.ERROR\_MESSAGE);*

*}*

*}*

***/\* A continuación se presenta el método de abrirConexionConElServidor para estar***

***enviando los datos desde el cliente\*/***

*public void abrirConexion(){*

*String ip = "127.0.0.1";*

*int puerto = 1025;*

*try{*

***//Establecemos la conexión con el servidor***

*cliente = new Socket(ip, puerto);*

***//Obtenemos los objetos para poder leer y escribir a través***

***//del socket***

*entrada = cliente.getInputStream();*

*salida = cliente.getOutputStream();*

*lector = new DataInputStream(entrada);*

*escritor = new DataOutputStream(salida);*

*}*

*catch (NumberFormatException nfe){*

***//Notificamos error de puerto***

*JOptionPane.showMessageDialog(this,"Numero de puerto incorrecto","Error",JOptionPane.ERROR\_MESSAGE);*

*}*

*catch (Exception e){*

***//Notificamos cualquier otro error***

*JOptionPane.showMessageDialog(this, "Error de conexión", "Error",*

*JOptionPane.ERROR\_MESSAGE);*

*}*

*}*

***// Se tiene el método para cerrar el socket previamente abierto.***

*public void cerrarSocket(){*

*String msg = null;*

*try{*

*leyendo = false;*

*if (cliente != null){*

***//Con esto se cierra la conexión con el servidor***

*cliente.close();*

*cliente = null;*

*}*

*}*

*catch (Exception e){*

*msg = "Error al intentar cerrar conexión";*

*JOptionPane.showMessageDialog(this, msg, "Error",*

*JOptionPane.ERROR\_MESSAGE);*

*}*

*}*

***/\*A continuación se presenta el método que envía datos, mediante un DataOutputStream \*/***

*public void enviarDatos(){*

*try{*

*escritor.writeUTF(txtNombreCliente.getText());*

*escritor.writeUTF(txtDireccion.getText());*

*escritor.writeUTF(txtTelefono.getText());*

*escritor.writeUTF(String.valueOf(pack));*

*txtAreaEstatus.setText("Pedido enviado al restaurant, esperando respuesta...");*

*}*

*catch (Exception e){*

*}*

*}*

*public void leerDatos(){*

*Thread t = null;*

***/\*Creamos un objeto para leer datos del cliente indefinidamente***

***a través de un hilo. La interface Runnable solamente declara una función***

***miembro denominada run, que han de definir las clases que implementen este interface***

***\*/***

*Runnable r = new Runnable(){*

***/\* Se declara entonces la función run, que implementa la interface***

*Runnable\*/*

*@Override*

*public void run(){*

*if (leyendo)*

*return;*

*leyendo = true;*

*while (leyendo){*

***/\*Mientras se está leyendo se toman los datos que se están***

***introduciendo, en la caja de texto txtAreaEstatus \*/***

*try{*

*//Se escribe lo que se recibe del servidor*

*txtAreaEstatus.setText("Estimado cliente "+txtNombreCliente.getText()+"\nhemos recibido su pedido Pack "+String.valueOf(pack)+"\nlo estaremos enviando a la dirección "+txtDireccion.getText()+lector.readUTF());*

*/****/ Se resetean los componentes de texto***

*txtNombreCliente.setText("");*

*txtDireccion.setText("");*

*txtTelefono.setText("");*

***// Cerramos la conexión***

*cerrarSocket();*

***// la volvemos abrir***

*abrirConexion();*

*pack=0;*

*}*

*catch (Exception e){*

*}*

*}*

*}*

*};*

*t = new Thread(r);*

***//Se inicializa el hilo de lectura***

*t.start();*

*}*

*}*

***Clase HamburguesaServidor***

*package hamburguesa;*

*import java.awt.event.WindowAdapter;*

*import java.awt.event.WindowEvent;*

*import java.io.DataInputStream;*

*import java.io.DataOutputStream;*

*import java.io.IOException;*

*import java.io.InputStream;*

*import java.io.OutputStream;*

*import java.net.ServerSocket;*

*import java.net.Socket;*

*import java.util.logging.Level;*

*import java.util.logging.Logger;*

*import javax.swing.JOptionPane;*

*public final class HamburguesaServidor extends javax.swing.JFrame{*

***// Declaración de variables para apariencia***

*javax.swing.JLabel lblNombreRepartidor;*

*javax.swing.JLabel lblTiempoEntrega;*

*javax.swing.JTextField txtNombreRepartidor;*

*javax.swing.JTextField txtTiempoEntrega;*

*javax.swing.JTextArea txtAreaEstatus;*

*javax.swing.JScrollPane ScrollEstatus;*

*javax.swing.JButton btnAsignarPedido;*

***//Socket Servidor que aceptara conexiones de clientes***

*ServerSocket serverSocket;*

***//Objeto que se comunicará con los clientes para enviar y recibir datos.***

*Socket cliente;*

***//Objetos que permitirán leer y escribir flujos de bits través del socket.***

*InputStream entrada;*

*OutputStream salida;*

***//Objetos que permitirán leer valores de algún tipo desde el socket.***

*DataInputStream lector;*

*DataOutputStream escritor;*

***//Esta variable sirve para mantener o cerrar el ciclo de lectura de datos***

***//desde el cliente***

*boolean leyendo;*

*public HamburguesaServidor(){*

*initComponents();*

*abrirSocket();*

*}*

*private void initComponents() {*

***/\* En esta sección se tienen los elementos que compondrán la apariencia grafica***

***de la aplicación, declaraciones y acomodos en la pantalla \*/***

*java.awt.GridBagConstraints gridBagConstraints;*

*lblNombreRepartidor = new javax.swing.JLabel();*

*lblTiempoEntrega = new javax.swing.JLabel();*

*txtNombreRepartidor = new javax.swing.JTextField();*

*txtTiempoEntrega= new javax.swing.JTextField();*

*txtAreaEstatus = new javax.swing.JTextArea();*

*ScrollEstatus = new javax.swing.JScrollPane();*

*btnAsignarPedido = new javax.swing.JButton();*

*addWindowListener(new WindowAdapter() {*

*@Override*

*public void windowClosing(WindowEvent e){*

*System.exit(0);*

*cerrarSocket();*

*}*

*});*

*setTitle("La Hamburguesa (Servidor)");*

*lblNombreRepartidor.setText("Nombre Repartidor:");*

*lblTiempoEntrega.setText("Tiempo de Entrega:");*

*txtNombreRepartidor.setEditable(false);*

*txtTiempoEntrega.setEditable(false);*

*txtAreaEstatus.setEditable(false);*

*btnAsignarPedido.setEnabled(false);*

*btnAsignarPedido.setText("Asignar Pedido");*

*btnAsignarPedido.addActionListener(new java.awt.event.ActionListener() {*

*@Override*

*public void actionPerformed(java.awt.event.ActionEvent evt) {*

*asignarPedido();*

*}*

*});*

*getContentPane().setLayout(new java.awt.GridBagLayout());*

*setBounds(250, 160, 500, 320);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(lblNombreRepartidor, gridBagConstraints);*

*lblTiempoEntrega.setPreferredSize(new java.awt.Dimension(80, 14));*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 0;*

*gridBagConstraints.gridy = 1;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(lblTiempoEntrega, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 1;*

*gridBagConstraints.gridy = 0;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 20, 5, 5);*

*getContentPane().add(txtNombreRepartidor, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 1;*

*gridBagConstraints.gridy = 1;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.insets = new java.awt.Insets(5, 20, 5, 5);*

*getContentPane().add(txtTiempoEntrega, gridBagConstraints);*

*txtAreaEstatus.setColumns(20);*

*txtAreaEstatus.setRows(5);*

*ScrollEstatus.setViewportView(txtAreaEstatus);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 0;*

*gridBagConstraints.gridy = 5;*

*gridBagConstraints.gridwidth = 2;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;*

*gridBagConstraints.weightx = 1.0;*

*gridBagConstraints.weighty = 1.0;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(ScrollEstatus, gridBagConstraints);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 0;*

*gridBagConstraints.gridy = 4;*

*gridBagConstraints.gridwidth = 4;*

*gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);*

*getContentPane().add(btnAsignarPedido, gridBagConstraints);*

*}*

***/\* A continuación se presenta el método de abrirSocket \*/***

*public void abrirSocket(){*

*String msg = null;*

*int puerto = 1025;*

*try {*

***//Se asigna ese Puerto al socket creado***

*serverSocket = new ServerSocket(puerto);*

*/****/Notificamos que estamos a la espera de una solicitud de parte del cliente.***

*msg ="Esperando nueva solicitud...";*

*txtAreaEstatus.setText(msg);*

***//Esperamos a que un cliente se conecte.***

*cliente = serverSocket.accept();*

***//Obtenemos los objetos para poder leer y escribir a través***

***//del socket***

*entrada = cliente.getInputStream();*

*salida = cliente.getOutputStream();*

*lector = new DataInputStream(entrada);*

*escritor = new DataOutputStream(salida);*

*leerDatosDelCliente();*

*}*

*catch (NumberFormatException nfe){*

***//Notificamos error de puerto***

*JOptionPane.showMessageDialog(this,"Numero de puerto incorrecto","Error",JOptionPane.ERROR\_MESSAGE);*

*}*

*catch (Exception e){*

***//Notificamos cualquier otro error***

*JOptionPane.showMessageDialog(this, "Error de conexión", "Error",*

*JOptionPane.ERROR\_MESSAGE);*

*}*

*}*

***// Inicia el método de cerrarSocket***

*public void cerrarSocket(){*

*String msg = null;*

*try{*

*leyendo = false;*

*/****/ Se está en espera de recibir comunicación por parte del cliente.***

*if (cliente != null){*

***//Con esto se cierra la conexión con el cliente.***

*cliente.close();*

***//Con esto se cierra el socket servidor, ya no atendemos más conexiones.***

*serverSocket.close();*

*cliente = null;*

*serverSocket = null;*

*}*

*}*

***/\* En el caso de que no se pueda cerrar el socket se captura la excepción y***

***se cierra el sistema \*/***

*catch (Exception e){*

*msg = "Error al intentar cerrar conexión";*

*JOptionPane.showMessageDialog(this, msg, "Error",JOptionPane.ERROR\_MESSAGE);*

*}*

*}*

***/\* El siguiente método es para leer los datos que el cliente envía hacia el***

***servidor \*/***

*public void leerDatosDelCliente(){*

*Thread t = null;*

***/\*Creamos un objeto para leer datos del cliente indefinidamente***

***a través de un hilo. La interface Runnable solamente declara una función***

***miembro denominada run, que han de definir las clases que implementen este interface***

***\*/***

*Runnable r = new Runnable(){*

***/\* Se declara entonces la función run, que implementa la interface***

*Runnable\*/*

*@Override*

*public void run(){*

*if (leyendo)*

*return;*

*leyendo = true;*

*while (leyendo){*

***/\*Mientras se está leyendo se toman los datos que se están***

***introduciendo, en la caja de texto txtAreaEstatus \*/***

*try{*

***//Se asignan los datos leídos a sus respectivos componentes***

*txtAreaEstatus.setText("Nombre Cliente: "+lector.readUTF()+"\nDirección de entrega: "+ lector.readUTF()+ "\nTeléfono: "+lector.readUTF()+"\nPack "+lector.readUTF());*

*if(entrada != InputStream.nullInputStream()){*

*txtNombreRepartidor.setEditable(true);*

*txtTiempoEntrega.setEditable(true);*

*btnAsignarPedido.setEnabled(true);*

*}*

*}*

*catch (Exception e){*

*}*

*}*

*}*

*};*

*t = new Thread(r);*

***//Se inicializa el hilo de lectura***

*t.start();*

*}*

***//Esta función se encarga asignar los datos, enviarlos y volver a quedar a la espera de otra conexión***

*public void asignarPedido() {*

*if("".equals(txtNombreRepartidor.getText()) || "".equals(txtTiempoEntrega.getText())){*

*JOptionPane.showMessageDialog(this, "Alguno de los campos esta vacio", "Error",JOptionPane.ERROR\_MESSAGE);*

*}else{*

*txtAreaEstatus.append("\nNombre repartidor: "+txtNombreRepartidor.getText());*

*txtAreaEstatus.append("\nTiempo de entrega: "+txtTiempoEntrega.getText());*

*txtAreaEstatus.append("\n\nPedido en proceso... ");*

*enviarDatos();*

*txtNombreRepartidor.setText("");*

*txtTiempoEntrega.setText("");*

*txtNombreRepartidor.setEditable(false);*

*txtTiempoEntrega.setEditable(false);*

*btnAsignarPedido.setEnabled(false);*

*try {*

*cliente.close();*

***//Esperamos a que un cliente se conecte.***

*cliente = serverSocket.accept();*

*entrada = cliente.getInputStream();*

*salida = cliente.getOutputStream();*

*lector = new DataInputStream(entrada);*

*escritor = new DataOutputStream(salida);*

*leerDatosDelCliente();*

*} catch (IOException ex) {*

*Logger.getLogger(HamburguesaServidor.class.getName()).log(Level.SEVERE, null, ex);*

*}*

*}*

*}*

***//Función encargada de Enviar los datos al cliente***

*public void enviarDatos(){*

*try{*

***//Se obtienen los datos de los componentes y se asignan a una variable String***

*String datos = "\nen "+ txtTiempoEntrega.getText()+" minutos, su repartidor es "+ txtNombreRepartidor.getText() + "\ngracias por su preferencia.";*

***//Se envían al cliente***

*escritor.writeUTF(datos);*

*}*

*catch (Exception e){*

*}*

*}*

*}*

# **Conclusión:**

La programación en red se refiere al uso de lenguajes d programación de computadoras para lograr el intercambio de recursos y la transferencia de información entre computadoras bajo la administración y coordinación de sistemas operativos, software de administración de red y protocolos de comunicación de red.

La arquitectura cliente-servidor es un modelo en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un programa cliente realiza peticiones a otro programa, el servidor, que le da respuesta.

La comunicación entre procesos a través de sockets se basa en la filosofía cliente servidor: un proceso en esta comunicación actuará de proceso servidor creando un socket cuyo nombre conocerá el proceso cliente, el cual podrá se comunicará con el servidor a través de la conexión con dicho socket nombrado.

En el modelo cliente servidor hay 2 roles en la interacción: por una parte, cliente que solicita servicio, con rol activo al iniciar la interacción, por otra parte el servidor que proporciona servicio y su rol es pasivo porque responde a petición de servicio.

Las desventajas del cliente/servidor puede ser un servidor "cuello de botella que genere problemas de escalabilidad o un servidor punto crítico de fallo o el mal aprovechamiento de recursos de máquinas cliente.

# **Bibliografía:**

* De la Torre, C., Zorrilla, U., Calvarro, J. y Ramos, M. A. 010). Guía de arquitectura N Capas orientada al dominio con .NET 4.J. Madrid: Krasis Press.
* IT.UC#M (2003). Trabajando con sockets TCP. http://www.it.uc3m.es/celeste/docencia/cr/2003/PracticaSockets TCP/ Recuperado
* Macedo, Y (2015). Arquitecturas cliente/servidor. Creación de Sockets Cliente Servidor. Recuperado de http://profesores.fi-b.unam.mx/yasmine/tema2\_tcp.pdf
* Pérez, F. (2021). Arquitectura de los Sistemas Distribuidos. Recuperado de https://laurel.datsi.fi.upm.es/\_media/docencia/asignaturas/sod/arquitecturas\_1pp.pdf
* Programmerclick (2019). Programación de redes Java-1: Conceptos básicos de redes. Recuperado de https://programmerclick.com/article/79611610595/