Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Introducción a la Programación y Computación 1
Sección N
Catedrático: Ing. Marlon Orellana
Tutor académico: Darwin Arévalo

Manual Técnico Proyecto Fase 1

Autores:

Jonatan David Reyna Monterroso	202002882
Gerber Emerson Ordoñez Tucubal	202004060
Gerson Rubén Quiroa del Cid	202000166
Johnny Whillman Aldana Osorio	201807499

Contenido

Requisitos del Sistema

Su computadora debe cumplir como mínimo los siguientes requerimientos:

Para Windows:

- Windows Vista SP2 (8u51 y superiores)
- Windows Server 2008 R2 SP1 (64 bits)
- Windows Server 2012 y 2012 R2 (64 bits)
- RAM: 128 MB
- Espacio en disco: 124 MB para JRE; 2 MB para Java Update
- Procesador: Mínimo Pentium 2 a 266 MHz
- Exploradores: Internet Explorer 9 y superior, Firefox

Mac OS X

- Mac con Intel que ejecuta Mac OS X 10.8.3+, 10.9+
- Privilegios de administrador para la instalación
- Explorador de 64 bits
- Se requiere un explorador

Linux

- Oracle Linux 7.x (64 bits)2
- (8u20 y superiores)
- Red Hat Enterprise Linux 7.x (64 bits)2
- (8u20 y superiores)
- Suse Linux Enterprise Server 12.x (64 bits)2
- (8u31 y superiores)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 15.10 (8u65 y superiores)

IMPORTANTE: Independientemente del sistema operativo delusuario, es necesario que previamente instale el JDK en su computadora.

Clase Main

En el Main se hace primero la leida de los archivos “json” o “bin” dependiendo de la configuración que se tenga. Creamos el objeto de la clase “Carga Masiva” para lograr la leída del archivo, y con un switch and case el programa lee la parte “load” de “config.json” y procede a cargar los archivos. Si no se encuentra ningún archivo este retornará un error.

```java
public class Main {
    public static void main(String[] args) {
        CargaMasiva cargaMasiva = new CargaMasiva();
        Login login = new Login();
        cargaMasiva.cargaConfig();

        switch (CargaMasiva.restaurant.getLoad()) {
            case "bin":
                cargaMasiva.cargaDatosBin();
                break;
            case "json":
                cargaMasiva.cargaDatosJson();
                break;
            default:
                System.out.println("No se han podido cargar los datos, verifique que la configuración sea correcta.");
                break;
        }
        login.login();
    }
```

Clase CargaMasiva

getContentofFile():
Con este método leemos el archivo deseado haciendo uso de “FileReader”y “BufferedReader”.

```java
public static String getContentOfFile(String pathname) {
    File archivo = null;
    FileReader fr = null;
    BufferedReader br = null;

    try {
        // Apertura del fichero y creacion de BufferedReader para poder
        // hacer una lectura comoda (disponer del metodo readLine()).
        archivo = new File(pathname);
        fr = new FileReader(archivo);
        br = new BufferedReader(fr);
        // Lectura del fichero
        String content = "";
        String linea;
        while ((linea = br.readLine()) != null) {
            content += linea + "\n";
        }
        return content;
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // En el finally cerramos el fichero, para asegurarnos
        // que se cierra tanto si todo va bien como si salta
        // una excepcion.
        try {
            if (null != fr) {
                fr.close();
            }
        } catch (Exception e2) {
            e2.printStackTrace();
        }
    }
    return "";
```

createFile():

Usamos este método para crear los archivos que necesitemos,

```
public static void createFile(String pathname, String data) {
    FileWriter flwriter = null;
    try {
        flwriter = new FileWriter(pathname, false); // True in
        BufferedWriter bfwriter = new BufferedWriter(flwriter)
        // Escribe los datos en el archivo
        bfwriter.write(data);
        bfwriter.close();

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (flwriter != null) {
            try {
                flwriter.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

cargaConfig():

Aquí leemos el archivo de configuración que el usuario quiera usar. Es de suma importancia porque de este archivo depende que se lean los archivos json o archivos bin.

```
public void cargaConfig() {
    String datos = getContentOfFile("config.json");
    restaurant = gson.fromJson(datos, Restaurante.class);
}
```

cargaDatosJson():

Aquí leemos los datos de los archivos json

```
public void cargaDatosJson() {
    try {
        String datos = getContentOfFile("users.json");
        Usuario[] usuarios = gson.fromJson(datos, Usuario[].class);
        users.addAll(Arrays.asList(usuarios));

        for (int i = 0; i < users.size() - 1; i++) {
            for (int j = i + 1; j < users.size(); j++) {
                if (users.get(i).getUsername().equals(users.get(j).getUsername())) {
                    String log = fechaHoraActual()
                            + "\t\tUSERNAME: El usuario " + users.get(j).getUsername() + " ya existe, se omitió el registro.\r\n";
                    Log.addToEndFile("errors.log", log);
                    users.remove(j);
                    break;
                }
            }
        }
        Log.addToEndFile("errors.log", "\r\n");

        datos = getContentOfFile("products.json");
        Producto[] productos = gson.fromJson(datos, Producto[].class);
        products.addAll(Arrays.asList(productos));

        repeticiones(products);

        datos = getContentOfFile("clients.json");
        Cliente[] clientes = gson.fromJson(datos, Cliente[].class);
        clients.addAll(Arrays.asList(clientes));

        repeticiones(clients);

        datos = getContentOfFile("invoices.json");
        Factura[] facturas = gson.fromJson(datos, Factura[].class);
        invoices.addAll(Arrays.asList(facturas));

        repeticiones(invoices);

        System.out.println("");
    } catch (JsonSyntaxException e) {
        System.out.println("Ocurrió un problema en la carga de archivos:(");
    }
}
```

cargaDatosBin():

En este método hacemos la leída de los datos en caso el usuario quiera cargar un archivo tipo bin.

```
public void cargaDatosBin() {
    try {
        users = (ArrayList<Usuario>) serializar.deserialize("Usuarios.ipcrm");
        products = (ArrayList<Producto>) serializar.deserialize("Productos.ipcrm");
        clients = (ArrayList<Cliente>) serializar.deserialize("Clientes.ipcrm");
        invoices = (ArrayList<Factura>) serializar.deserialize("Facturas.ipcrm");
        System.out.println("");
    } catch (Exception e) {

    }
}
```

repeticiones():

En este método buscamos los registros que están repetidos y en caso se encuentren unos, se lanza un mensaje de error al log.

```
public void repeticiones(ArrayList lista) {
    ArrayList<Objeto> listaa = lista;
    for (int i = 0; i < listaa.size() - 1; i++) {
        for (int j = i + 1; j < listaa.size(); j++) {
            if (listaa.get(i).getId() == listaa.get(j).getId()) {
                String log = fechaHoraActuales
                        + "\t\tINVOICES: El id " + listaa.get(j).getId() + " ya existe, se omitió el registro.\r\n";
                Log.addToEndFile("errors.log", log);
                listaa.remove(j);
                break;
            }
        }
    }
    Log.addToEndFile("errors.log", "\r\n");
}
```

guardarDatosJson():

Con este método usamos la serialización para mantener la prevalencia de datos y los guardamos en un archivo json.

```
public void guardarDatosJson() {
    try {
        String datos = prettyGson.toJson(users);
        createFile("users.json", datos);

        datos = prettyGson.toJson(products);
        createFile("products.json", datos);

        datos = prettyGson.toJson(clients);
        createFile("clients.json", datos);

        datos = prettyGson.toJson(invoices);
        createFile("invoices.json", datos);

        System.out.println("Se han guardado los cambios! :D");

    } catch (JsonIOException e) {
        System.out.println("Ocurrió un error. :(");
    }
}
```

guardarDatosBin():

Con este método usamos la serialización para mantener la prevalencia dedatos y los guardamos en un archivo Bin.

```
public void guardarDatosBin() {
    try {
        serializar.serialize("Usuarios.ipcrm", users); //serialización de usuarios
        serializar.serialize("Productos.ipcrm", products); //serialización de productos
        serializar.serialize("Clientes.ipcrm", clients); //serialización de clientes
        serializar.serialize("Facturas.ipcrm", invoices); //serialización de facturas
        System.out.println("Se han guardado los cambios! :D");
    } catch (Exception e) {
        System.out.println("Ocurrió un error. :(");
    }
}
```

Clase VentanaLogin

En esta clase creamos la ventana que maneja el ingreso de los usuarios a la aplicación, con el uso de JPasswordField hacemos que la contraseña ingresada no sea visible para el usuario. Llamamos a los métodos “initComponentes” y estética.

```
public class VentanaLogin extends JFrame {

    public VentanaLogin() {
        //Inicializando la ventana y dándole dimensiones
        VentanaLoginP ventanaLoginP = new VentanaLoginP();
        this.setBounds(400, 170, 350, 230);
        this.setTitle("Login");
        this.add(ventanaLoginP);
    }
}

class VentanaLoginP extends JPanel implements ActionListener {

    JLabel titulo, usuario, contraseña;
    JTextField usuarioT;
    JPasswordField contraseñaT;
    JButton aceptar;
    Login login;

    public VentanaLoginP() {
        login = new Login();
        initComponents();
        estetica();
    }
```

```java
@Override
public void actionPerformed(ActionEvent e) {
    String username = usuarioT.getText();
    String pass = String.valueOf(contraseñaT.getPassword());

    if (username.equals("") || pass.equals("")) {
        String m = "Usuario o contraseña sin llenar :(";
        JOptionPane.showMessageDialog(this, m, "Error", 2);
    } else {
        login.login(username, pass);
    }
}
```

InitComponentes()

Inicializamos los componentes y se añaden a la ventana

```java
public void initComponents() {
    titulo = new JLabel("Bienvenido");
    this.add(titulo);

    usuario = new JLabel("Usuario: ");
    this.add(usuario);
    usuarioT = new JTextField("");
    this.add(usuarioT);

    contraseña = new JLabel("Contraseña: ");
    this.add(contraseña);
    contraseñaT = new JPasswordField("");
    this.add(contraseñaT);

    aceptar = new JButton("Aceptar");
    this.add(aceptar);
    aceptar.addActionListener(this);
    this.setBackground(new Color(255, 128, 0));

}
```

Estética()

Se setean los bounds de los componentes que inicializamos anteriormente dándole dimensiones, así como dándole una Font personalizada al título.

```
public void estetica() {
    this.setLayout(null); //layout null para poder poner los componer

    titulo.setBounds(100, 10, 500, 35);
    titulo.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 30));

    usuario.setBounds(50, 60, 100, 25);
    usuarioT.setBounds(130, 63, 150, 18);

    contraseña.setBounds(50, 105, 100, 25);
    contraseñaT.setBounds(130, 108, 150, 18);

    aceptar.setBounds(130, 150, 100, 30);

}
```

Clase Ventana Menu

Esta es la ventana principal que el usuario ve, desde aquí puede acceder a las diferentes opciones que el programa ofrece, se crean los botones acceder a la información del restaurante, usuarios, productos, clientes, facturas y guardar cambios.

```
public class VentanaMenu extends JFrame {

    public VentanaMenu() {
        //Inicializando la ventana y dándole dimensiones
        VentanaMenuP ventanaMenuP = new VentanaMenuP();
        this.setBounds(300, 100, 750, 450);
        this.setTitle("Menú principal");
        this.add(ventanaMenuP);
    }
}

class VentanaMenuP extends JPanel implements ActionListener {

    JLabel titulo;
    JButton infoRest, usuario, producto, cliente, factura, gCambios;
    public static VentanaInfoRestaurante ventanaInfoRestaurante;
    public static VentanaUsuario ventanaUsuario;
    public static VentanaProducto ventanaProducto;
    public static VentanaClientes ventanaClientes;
    public static VentanaFactura ventanaFactura;
    public static JLabel name, adress, phone, load;
    public static String usuarioA = CargaMasiva.users.get(Login.usuario).getUsername();
```

InitComponentes()

Se inicializan los botones de acceder a la información del restaurante, usuarios, productos, clientes, facturas y guardar cambios así como los labels con la información del restaurante cuando el usuario la quiere mostrar.

```
public void initComponents() {
    titulo = new JLabel("¿Qué deseas hacer hoy "
            + usuarioA + "?");
    this.add(titulo);

    infoRest = new JButton("Información del restaurante");
    this.add(infoRest);
    infoRest.addActionListener(this);

    usuario = new JButton("Menú Usuarios");
    this.add(usuario);
    usuario.addActionListener(this);

    producto = new JButton("Menú Productos");
    this.add(producto);
    producto.addActionListener(this);

    cliente = new JButton("Menú Clientes");
    this.add(cliente);
    cliente.addActionListener(this);

    factura = new JButton("Menú Facturas");
    this.add(factura);
    factura.addActionListener(this);

    gCambios = new JButton("Guardar Cambios");
    this.add(gCambios);
    gCambios.addActionListener(this);

    //datos del restaurante
    name = new JLabel("");
    this.add(name);
    adress = new JLabel("");
    this.add(adress);
    phone = new JLabel("");
    this.add(phone);
    load = new JLabel("");
    this.add(load);
```

Estetica()

Se les da los bordes y tamaños a los botones y fuente a los labels.

```
private void estetica() {
    this.setLayout(null); //layout null para poder poner los componente

    titulo.setBounds(110, 10, 500, 35);
    titulo.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 30));

    infoRest.setBounds(30, 90, 200, 35);
    usuario.setBounds(260, 90, 200, 35);
    producto.setBounds(490, 90, 200, 35);

    cliente.setBounds(30, 170, 200, 35);
    factura.setBounds(260, 170, 200, 35);
    gCambios.setBounds(490, 170, 200, 35);
    name.setBounds(100, 250, 400, 20);
    name.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 15));
    adress.setBounds(100, 280, 400, 20);
    adress.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 15));
    phone.setBounds(100, 310, 400, 20);
    phone.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 15));
    load.setBounds(100, 340, 400, 20);
    load.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 15));

    this.setBackground(new Color(153, 153, 255));
}
```

Evento Botón

Con un if y else if, se elige cual ventana se abre dependiendo del botón que haya presionado el usuario.

```
public void actionPerformed(ActionEvent e) {
    JButton botonP = (JButton) e.getSource();

    if (botonP == infoRest) {
        ventanaInfoRestaurante.setVisible(true);
        Login.ventanaMenu.dispose();
        ventanaInfoRestaurante.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    } else if (botonP == usuario) {
        ventanaUsuario = new VentanaUsuario();
        Login.ventanaMenu.dispose();
        ventanaUsuario.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    } else if (botonP == producto) {
        ventanaProducto = new VentanaProducto();
        Login.ventanaMenu.dispose();
        ventanaProducto.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    } else if (botonP == cliente) {
        ventanaClientes = new VentanaClientes();
        Login.ventanaMenu.dispose();
        ventanaClientes.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    } else if (botonP == factura) {
        ventanaFactura = new VentanaFactura();
        Login.ventanaMenu.dispose();
        ventanaFactura.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    } else if (botonP == gCambios) {
        try {
            CargaMasiva.guardarDatosJson();
            String m = "Se han guardado los cambios con éxito :D.";
            JOptionPane.showMessageDialog(this, m, "Guardar Cambios", 1);
        } catch (JsonIOException ex) {
            String m = "Ocurrió un error. :(";
            JOptionPane.showMessageDialog(this, m, "Guardar Cambios", 1);
        }
    }
```

Clase Ventana Info Restaurante

En esta clase creamos la ventana donde se maneja la información del restaurante.

```
public class VentanaInfoRestaurante extends JFrame {

    public VentanaInfoRestaurante() {
        //Inicializando la ventana y dándole dimensiones
        VentanaInfoRestauranteP ventanaInfoRestaurante = new VentanaInfoRestauranteP();
        this.setBounds(300, 100, 500, 200);
        this.setTitle("Restaurante");
        this.add(ventanaInfoRestaurante);
        this.addWindowListener(ventanaInfoRestaurante);
    }
}

class VentanaInfoRestauranteP extends JPanel implements WindowListener, ActionListener {

    JLabel titulo;
    JButton editar, mostrar;

    public VentanaInfoRestauranteP() {
        initComponents();
        estetica();
    }
```

InitComponentes()

Creamos los botones para editar la información del restaurante y el botón para mostrar los datos del restaurante en la ventana principal.

```
public void initComponents() {
    titulo = new JLabel("Información del Restaurante");
    this.add(titulo);

    editar = new JButton("Editar");
    this.add(editar);
    editar.addActionListener(this);

    mostrar = new JButton("Mostrar datos");
    this.add(mostrar);
    mostrar.addActionListener(this);

}
```

Estetica()

Seteamos los bounds de los botones y labels

```
public void estetica() {
    this.setLayout(null); //layout null para poder poner los componentes

    titulo.setBounds(70, 10, 500, 35);
    titulo.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 30));

    editar.setBounds(100, 60, 100, 35);
    mostrar.setBounds(250, 60, 150, 35);

    this.setBackground(new Color(204, 204, 0));
}
```

Evento Window Closed

Este evento sucede cuando se cierra la ventana, lo cual hace que regrese a la ventana anterior.

```
@Override
public void windowClosed(WindowEvent e) {
    //Aquí volvemos a mostrar la ventana anterior que se cerró
    Login.ventanaMenu.setVisible(true);
}
```

Clase Ventana Usuario

Desde esta ventana, el usuario con ayuda de una tabla usando el componente de swing Jtable puede ver los usuarios que el programa tenga.

```
public class VentanaUsuario extends JFrame {

    public VentanaUsuario() {
        //Inicializando la ventana y dándole dimensiones
        VentanaUsuarioP ventanaUsuarioP = new VentanaUsuarioP();
        this.setBounds(300, 100, 500, 450);
        this.setTitle("Menú usuario");
        this.setVisible(true);
        this.add(ventanaUsuarioP);
        this.addWindowListener(ventanaUsuarioP);
    }
}

class VentanaUsuarioP extends JPanel implements ActionListener, WindowListener, MouseLis

    static DefaultTableModel modelo;
    static JTable tabla;
    Object[][] data;
    String[] columnNames = {"Usuario", "Contraseña", "Editar", "Eliminar"};
    JLabel titulo;
    JButton crear;
    static JButton editar, eliminar;

    public VentanaUsuarioP() {
        initComponents();
        tabla();
    }
```

InitComponentes()

Se crean los botones y labels que se ven en la ventana

```
public void initComponents() {
    this.setLayout(null); //layout null para poder poner los componentes donc
    this.setBackground(new Color(204, 204, 0));

    titulo = new JLabel("Menú Usuario");
    titulo.setBounds(120, 10, 500, 35);
    titulo.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 30));
    this.add(titulo);

    crear = new JButton("Crear usuario");
    crear.setBounds(50, 370, 120, 30);
    this.add(crear);
    crear.addActionListener(this);

    editar = new JButton("Editar");
    editar.setName("m");
    editar.addActionListener(this);
    eliminar = new JButton("Eliminar");
    editar.setName("e");
    eliminar.addActionListener(this);

}
```

Método Tabla

Se crea una tabla y se le mandan los datos de los usuarios. Junto a los botones eliminar y editar.

```
public void tabla() {
    tabla = new JTable();
    data = data2D();
    tabla.setDefaultRenderer(Object.class, new Render());
    modelo = new DefaultTableModel(data, columnNames) {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    tabla.setModel(modelo);
    tabla.setRowHeight(20);
    tabla.addMouseListener(this);
    JScrollPane scrollPane = new JScrollPane(tabla);
    scrollPane.setPreferredSize(new Dimension(380, 280));
    JPanel panel = new JPanel();
    panel.setBounds(20, 70, 430, 280);
    panel.add(scrollPane);
    this.add(panel, BorderLayout.CENTER);
}

public static Object[][] data2D() {
    Object[][] data = new Object[CargaMasiva.users.size()][4];
    for (int i = 0; i < CargaMasiva.users.size(); i++) {
        data[i][0] = CargaMasiva.users.get(i).getUsername();
        data[i][1] = CargaMasiva.users.get(i).getPassword();
        data[i][2] = editar;
        data[i][3] = eliminar;
    }
    if (modelo != null) {
        modelo = (DefaultTableModel) tabla.getModel();
        tabla.setModel(modelo);
    }
    return data;
}
```

Evento Botón Crear

Con este botón abrimos la ventana crear que se llama desde la clase “VentanaAñadir” y se cierra la ventana anterior

```
@Override
public void actionPerformed(ActionEvent e) {
    JButton botonP = (JButton) e.getSource();
    if (botonP == crear) {
        System.out.println("crear");
        VentanaAñadir ventanaAñadir = new VentanaAñadir(CargaMasiva.users, "usuario");
        VentanaMenuP.ventanaUsuario.setVisible(false);
        ventanaAñadir.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

Evento Boton Eliminar y Editar

Con este botón se eliminan los datos haciendo uso de la función remove, y se guarda la eliminación en el log, también se trabaja en la edición de usuarios donde llamamos a la clase “VentanaEditar” para poder editar al usuario que uno quiera.

```
erride
lic void mouseClicked(MouseEvent e) {
 int fila = tabla.rowAtPoint(e.getPoint());
 int columna = tabla.columnAtPoint(e.getPoint());

 if (columna == 3) {  //columna eliminar
     System.out.println(fila + " eliminar");
     logEliminacionU(fila);
     CargaMasiva.users.remove(fila);
     VentanaMenuP.ventanaUsuario.setVisible(false);
     VentanaMenuP.ventanaUsuario = new VentanaUsuario();
     VentanaMenuP.ventanaUsuario.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
 } else if (columna == 2) { //columna editar
     System.out.println(fila + " editar");
     VentanaEditar ventanaEditar = new VentanaEditar(CargaMasiva.users, fila, "usuario");
     VentanaMenuP.ventanaUsuario.setVisible(false);
     ventanaEditar.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
 }
```

Clase Ventana Clientes

En esta clase manejamos los datos de los clientes, asi como las funciones de eliminar, editar y crear clientes.

```
public class VentanaClientes extends JFrame {

    public VentanaClientes() {
        //Inicializando la ventana y dándole dimensiones
        VentanaClientesP ventanaClientesP = new VentanaClientesP();
        this.setBounds(300, 100, 630, 450);
        this.setTitle("Menú usuario");
        this.setVisible(true);
        this.add(ventanaClientesP);
        this.addWindowListener(ventanaClientesP);
    }
}

class VentanaClientesP extends JPanel implements ActionListener, MouseListener, WindowListener {

    static DefaultTableModel modelo;
    static JTable tabla;
    Object[][] data;
    String[] columnNames = {"Id", "Nombre", "Dirección", "Teléfono", "NIT", "Editar", "Eliminar"};
    JLabel titulo;
    JButton crear;
    static JButton editar, eliminar;

    public VentanaClientesP() {
        initComponents();
        tabla();
    }
```

InitComponentes()

Añadimos los componentes a la ventana y les damos dimensiones.

```
public void initComponents() {
    this.setLayout(null); //layout null para poder poner los componente
    this.setBackground(new Color(204, 204, 0));

    titulo = new JLabel("Menú Clientes");
    titulo.setBounds(180, 10, 500, 35);
    titulo.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 30));
    this.add(titulo);

    crear = new JButton("Crear Cliente");
    crear.setBounds(50, 370, 150, 30);
    this.add(crear);
    crear.addActionListener(this);

    editar = new JButton("Editar");
    editar.setName("m");
    editar.addActionListener(this);
    eliminar = new JButton("Eliminar");
    editar.setName("e");
    eliminar.addActionListener(this);
}
```

Método Tabla

Creamos la tabla usando JTable, le damos dimensiones y un scroll para poder ver toda la tabla y le damos dimensiones, asi como le damos los datos de los clientes a partir de la carga masiva.

```
public void tabla() {
    tabla = new JTable();
    data = data2D();
    tabla.setDefaultRenderer(Object.class, new Render());
    modelo = new DefaultTableModel(data, columnNames) {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    tabla.setModel(modelo);
    tabla.setRowHeight(20);
    tabla.addMouseListener(this);
    JScrollPane scrollPane = new JScrollPane(tabla);
    scrollPane.setPreferredSize(new Dimension(570, 280));
    JPanel panel = new JPanel();
    panel.setBounds(20, 70, 570, 280);
    panel.add(scrollPane);
    this.add(panel, BorderLayout.CENTER);
}

public static Object[][] data2D() {
    Object[][] data = new Object[CargaMasiva.clients.size()][7];
    for (int i = 0; i < CargaMasiva.clients.size(); i++) {
        data[i][0] = CargaMasiva.clients.get(i).getId();
        data[i][1] = CargaMasiva.clients.get(i).getName();
        data[i][2] = CargaMasiva.clients.get(i).getAddress();
        data[i][3] = CargaMasiva.clients.get(i).getPhone();
        data[i][4] = CargaMasiva.clients.get(i).getNit();
        data[i][5] = editar;
        data[i][6] = eliminar;
    }
    if (modelo != null) {
        modelo = (DefaultTableModel) tabla.getModel();
        tabla.setModel(modelo);
    }
    return data;
}
```

Evento Crear

Llamamos a la ventana Añadir para asi crear un cliente.

```
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == crear) {
        System.out.println("crear");
        VentanaAñadir ventanaAñadir = new VentanaAñadir(CargaMasiva.clients, "clientes");
        VentanaMenuP.ventanaClientes.setVisible(false);
        ventanaAñadir.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

Evento Eliminar y Editar

Con este botón se eliminan los datos haciendo uso de la función remove, y se guarda la eliminación en el log, también se trabaja en la edición de usuarios donde llamamos a la clase “VentanaEditar” para poder editar al usuario que uno quiera.

```
@Override
public void mouseClicked(MouseEvent e) {
    int fila = tabla.rowAtPoint(e.getPoint());
    int columna = tabla.columnAtPoint(e.getPoint());

    if (columna == 6) {  //columna eliminar
        System.out.println(fila + " eliminar");
        logEliminacionC(fila);
        CargaMasiva.clients.remove(fila);
        VentanaMenuP.ventanaClientes.setVisible(false);
        VentanaMenuP.ventanaClientes = new VentanaClientes();
        VentanaMenuP.ventanaClientes.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    } else if (columna == 5) { //columna editar
        System.out.println(fila + " editar");
        VentanaEditar ventanaEditar = new VentanaEditar(CargaMasiva.clients, fila, "cliente");
        VentanaMenuP.ventanaClientes.setVisible(false);
        ventanaEditar.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

LogEliminacion Clientes

De haber eliminado a un cliente, se guarda el procedimiento en el log.

```
public void logEliminacionC(int fila) {
    String log = fechaHoraActuales
            + "\t\t" + VentanaMenuP.usuarioA
            + ": Eliminó el cliente " + CargaMasiva.clients.get(fila).getId() + ".\r\n";
    Log.addToEndFile("log.log", log);
}
```

Clase Ventana Factura

En esta clase creamos la ventana que le muestra los datos de facturas a los usuarios por medio de una tabla.

```
public class VentanaFactura extends JFrame {

    public VentanaFactura() {
        //Inicializando la ventana y dándole dimensiones
        VentanaFacturaP ventanaFacturaP = new VentanaFacturaP();
        this.setBounds(300, 100, 750, 450);
        this.setTitle("Menú Factura");
        this.setVisible(true);
        this.add(ventanaFacturaP);
        this.addWindowListener(ventanaFacturaP);
    }
}

class VentanaFacturaP extends JPanel implements ActionListener, WindowListener {

    static DefaultTableModel modelo;
    static JTable tabla;
    Object[][] data;
    String[] columnNames = {"Id", "Cliente", "Fecha", "Productos"};
    JLabel titulo;
    JButton crear;
    static JButton editar, eliminar;

    public VentanaFacturaP() {
        initComponents();
        tabla();
    }
```

InitComponentes()

```
public void initComponents() {
    this.setLayout(null); //layout null para poder poner los componen
    this.setBackground(new Color(204, 204, 0));

    titulo = new JLabel("Menú Facturas");
    titulo.setBounds(180, 10, 500, 35);
    titulo.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 30));
    this.add(titulo);

    crear = new JButton("Crear Factura");
    crear.setBounds(50, 370, 150, 30);
    this.add(crear);
    crear.addActionListener(this);

}
```

Método Tabla

Creamos la tabla y ahora le mandamos los datos de la clase Invoices que controla las facturas

```
public void tabla() {
    tabla = new JTable();
    data = data2D();
    modelo = new DefaultTableModel(data, columnNames) {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    tabla.setModel(modelo);
    tabla.setRowHeight(20);
    JScrollPane scrollPane = new JScrollPane(tabla);
    scrollPane.setPreferredSize(new Dimension(700, 280));
    JPanel panel = new JPanel();
    panel.setBounds(20, 70, 700, 280);
    panel.add(scrollPane);
    this.add(panel, BorderLayout.CENTER);
}

public static Object[][] data2D() {
    Object[][] data = new Object[CargaMasiva.invoices.size()][4];
    for (int i = 0; i < CargaMasiva.invoices.size(); i++) {
        data[i][0] = CargaMasiva.invoices.get(i).getId();
        data[i][1] = CargaMasiva.invoices.get(i).getClient();
        data[i][2] = CargaMasiva.invoices.get(i).getDate();
        data[i][3] = CargaMasiva.invoices.get(i).getProducts();
    }
    if (modelo != null) {
        modelo = (DefaultTableModel) tabla.getModel();
        tabla.setModel(modelo);
    }
    return data;
}
```

Evento Crear

```
@Override
public void actionPerformed(ActionEvent e) {
   if (e.getSource() == crear) {
        System.out.println("crear");
        VentanaAñadir ventanaAñadir = new VentanaAñadir(CargaMasiva.invoices, "factura");
        VentanaMenuP.ventanaFactura.setVisible(false);
        ventanaAñadir.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

Clase Ventana Producto

En esta clase se manejan los datos de los productos, asi como la creación, edición y eliminación de los productos.

```
public class VentanaProducto extends JFrame {

    public VentanaProducto() {
        //Inicializando la ventana y dándole dimensiones
        VentanaProductoP ventanaProductoP = new VentanaProductoP();
        this.setBounds(300, 100, 630, 450);
        this.setTitle("Menú usuario");
        this.setVisible(true);
        this.add(ventanaProductoP);
        this.addWindowListener(ventanaProductoP);
    }
}

class VentanaProductoP extends JPanel implements ActionListener, WindowListener, MouseListener {

    static DefaultTableModel modelo;
    static JTable tabla;
    Object[][] data;
    String[] columnNames = {"Id", "Nombre", "Descripción", "Costo", "Precio", "Ingredientes", "Editar", "Eliminar"};
    JLabel titulo;
    JButton crear;
    static JButton editar, eliminar;

    public VentanaProductoP() {
        initComponents();
        tabla();
    }
```

InitComponentes()

```
public void initComponents() {
    this.setLayout(null); //layout null para poder poner los componente
    this.setBackground(new Color(204, 204, 0));

    titulo = new JLabel("Menú Producto");
    titulo.setBounds(180, 10, 500, 35);
    titulo.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 30));
    this.add(titulo);

    crear = new JButton("Crear Producto");
    crear.setBounds(50, 370, 150, 30);
    this.add(crear);
    crear.addActionListener(this);

    editar = new JButton("Editar");
    editar.setName("m");
    editar.addActionListener(this);
    eliminar = new JButton("Eliminar");
    editar.setName("e");
    eliminar.addActionListener(this);

}
```

Método Tabla

Creamos la tabla, le damos dimensiones y le mandamos los datos de la clase productos que maneja los atributos de estos, asi como añadimos los botones editar y eliminar.

```
public void tabla() {
    tabla = new JTable();
    data = data2D();
    tabla.setDefaultRenderer(Object.class, new Render());
    modelo = new DefaultTableModel(data, columnNames) {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    tabla.setModel(modelo);
    tabla.setRowHeight(20);
    tabla.addMouseListener(this);
    JScrollPane scrollPane = new JScrollPane(tabla);
    scrollPane.setPreferredSize(new Dimension(570, 280));
    JPanel panel = new JPanel();
    panel.setBounds(20, 70, 570, 280);
    panel.add(scrollPane);
    this.add(panel, BorderLayout.CENTER);
}

public static Object[][] data2D() {
    Object[][] data = new Object[CargaMasiva.products.size()][8];
    for (int i = 0; i < CargaMasiva.products.size(); i++) {
        data[i][0] = CargaMasiva.products.get(i).getId();
        data[i][1] = CargaMasiva.products.get(i).getName();
        data[i][2] = CargaMasiva.products.get(i).getDescription();
        data[i][3] = CargaMasiva.products.get(i).getCost();
        data[i][4] = CargaMasiva.products.get(i).getPrice();
        data[i][5] = CargaMasiva.products.get(i).getIngredients();
        data[i][6] = editar;
        data[i][7] = eliminar;
    }
    if (modelo != null) {
        modelo = (DefaultTableModel) tabla.getModel();
        tabla.setModel(modelo);
    }
    return data;
}
```

Evento Botón Crear

```
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == crear) {
        System.out.println("crear");
        VentanaAñadir ventanaAñadir = new VentanaAñadir(CargaMasiva.products, "producto");
        VentanaMenuP.ventanaProducto.setVisible(false);
        ventanaAñadir.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

Evento Botón Editar y Eliminar

```
@Override
public void mouseClicked(MouseEvent e) {
    int fila = tabla.rowAtPoint(e.getPoint());
    int columna = tabla.columnAtPoint(e.getPoint());

    if (columna == 7) {  //columna eliminar
        System.out.println(fila + " eliminar");
        logEliminacionP(fila);
        CargaMasiva.products.remove(fila);
        VentanaMenuP.ventanaProducto.setVisible(false);
        VentanaMenuP.ventanaProducto = new VentanaProducto();
        VentanaMenuP.ventanaProducto.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    } else if (columna == 6) { //columna editar
        System.out.println(fila + " editar");
        VentanaEditar ventanaEditar = new VentanaEditar(CargaMasiva.products, fila, "producto");
        VentanaMenuP.ventanaProducto.setVisible(false);
        ventanaEditar.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

LogEliminacion Producto

En caso el usuario haya eliminado un producto, este se manda al Log con la siguiente función.

```
public void logEliminacionP(int fila) {
    String log = fechaHoraActuales
            + "\t\t" + VentanaMenuP.usuario2
            + ": Eliminó el producto " + CargaMasiva.products.get(fila).getId() + ".\r\n";
    Log.addToEndFile("log.log", log);
}
```

Clase Ventana Añadir

Esta clase se usa para crear la ventana que el botón Crear llama desde otras ventanas. Inicializamos los Componentes que se usaron.

```
public class VentanaAñadir extends JFrame {

    public VentanaAñadir(ArrayList lista, String tipo) {
        VentanaAñadirP ventanaAñadirP = new VentanaAñadirP(lista, tipo);
        initComponents(ventanaAñadirP);
    }

    public void initComponents(VentanaAñadirP ventanaAñadirP) {
        //Inicializando la ventana y dándole dimensiones
        this.setBounds(300, 100, 550, 320);
        this.setTitle("Editar");
        this.setVisible(true);
        this.add(ventanaAñadirP);
        this.addWindowListener(ventanaAñadirP);
    }
}
```

Creamos los componentes que se usan, jLabels, textFields y botones para cada una de las opciones de creación.

```
class VentanaAñadirP extends JPanel implements ActionListener, WindowListener {

    //CREACIÓN COMUNES
    JLabel titulo, nameL, idL;
    JTextField name, id;
    JButton crearU, crearP, crearI, crearC, crearF;
    boolean u, p, c, f, repeticion;
    //CREACIÓN USUARIO
    JTextField password, cPassword;
    ArrayList<Usuario> users;
    //CREACIÓN PRODUCTO
    JTextField description, cost, price;
    ArrayList<Producto> products;
    JButton crearIngrediente;
    //CREACIÓN INGREDIENTE
    JTextField quantity, units;
    ArrayList<Ingrediente> ingredients = new ArrayList<>();
    //CRECIÓN CLIENTE
    JTextField adress, phone, nit;
    ArrayList<Cliente> clients;
    //CREACIÓN FACTURA
    JTextField client, date;
    ArrayList<Factura> invoices;
    JButton aProducto;

    public VentanaAñadirP(ArrayList lista, String tipo) {
        estetica();
        if (tipo.equals("usuario")) {
            claseU(lista);
        } else if (tipo.equals("producto")) {
            claseP(lista);
        } else if (tipo.equals("ingrediente")) {
            claseI(lista);
        } else if (tipo.equals("clientes")) {
            claseC(lista);
        } else if (tipo.equals("factura")) {
            claseF(lista);
        }
    }
```

Creación de entradas

Método usado para la creación de la ventana de Crear Usuario, Producto, Ingrediente, Cliente y Factura. Se usa un constructor para añadir a cada uno de los datos, con labels y textfields correspondientes a la clase que se quiera añadir, por ejemplo, la creación de usuarios tiene tres text fields, uno para el nombre, y dos para la contraseña.

```java
public void claseU(ArrayList users) {
    //Constructor para añadir un usuario
    this.users = users;
    u = true;
    titulo = new JLabel("CREACIÓN DE USUARIOS");
    titulo.setBounds(70, 10, 500, 35);
    titulo.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 30));
    this.add(titulo);

    nameL = new JLabel("Nombre de usuario:");
    nameL.setBounds(30, 75, 150, 20);
    this.add(nameL);
    name = new JTextField("");
    name.setBounds(180, 75, 200, 20);
    name.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 15));
    this.add(name);

    JLabel passwordL = new JLabel("Contraseña de usuario:");
    passwordL.setBounds(30, 115, 150, 20);
    this.add(passwordL);
    password = new JTextField("");
    password.setBounds(180, 115, 200, 20);
    password.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 15));
    this.add(password);

    JLabel cPasswordL = new JLabel("Confirmar contraseña:");
    cPasswordL.setBounds(30, 155, 150, 20);
    this.add(cPasswordL);
    cPassword = new JTextField("");
    cPassword.setBounds(180, 155, 200, 20);
    cPassword.setFont(new Font("Bienvenido", Font.ROMAN_BASELINE, 15));
    this.add(cPassword);

    crearU = new JButton("Crear");
    crearU.setBounds(400, 115, 100, 30);
    this.add(crearU);
    crearU.addActionListener(this);
}
```

Métodos Añadir

Con estos métodos se manejan los nuevos datos que se crean. En caso de la imagen, se ve la creación de un usuario, donde se ve si las dos contraseñas son iguales y se añade este usuario nuevo al Array de usuarios. Si el usuario es creado con éxito se muestra un mensaje, de lo contrario se muestra un mensaje de error. Estos métodos engloban la creación de Productos, Usuarios, Facturas, Clientes e Ingredientes.

```java
public void añadirU() {
    try {
        repeticion = false;
        if (password.getText().equals(cPassword.getText())) {
            Usuario usuarioN = new Usuario(name.getText(), password.getText());
            for (int i = 0; i < CargaMasiva.users.size(); i++) {
                if (usuarioN.getUsername().equals(CargaMasiva.users.get(i).getUsername())) {
                    repeticion = true;
                }
            }
            if (repeticion == false) {
                CargaMasiva.users.add(usuarioN);
                log(usuarioN, "usuario");
                String m = "Se ha creado el usuario con éxito :D";
                JOptionPane.showMessageDialog(this, m, "Creación Usuario", 1);
            } else {
                String m = "Este usuario ya existe, intente otro.";
                JOptionPane.showMessageDialog(this, m, "Creación Usuario", 1);
            }
        } else {
            String m = "Las contraseñas no coinciden";
            JOptionPane.showMessageDialog(this, m, "Creación Usuario", 2);
        }
    } catch (NumberFormatException e) {
        String m = "Se ha introducido una letra donde debería ir un número.";
        JOptionPane.showMessageDialog(this, m, "Creación Usuario", 1);
    }
}
```

Serialización

Se necesita mantener los cambios realizados en el tiempo. Para que cada vez que se inicie el programa, los datos se mantengan actualizados. Para ello, los objetos del sistema se serializarán a JSON.

Para que un programa Java pueda convertir un objeto en un montón de bytes y pueda luego recuperarlo, el objeto necesita ser Serializable. Al poder convertir el objeto a bytes, ese objeto se puede enviar a través de red, guardarlo en un fichero, y después reconstruirlo al otro lado de la red, leerlo del fichero, etc.

Esto se logra con la siguiente clase:

```java
public class Serializacion {

    public void serialize(String pathname, Object object) {
        // Serializar un objeto
        try {
            ObjectOutputStream objectOutputStream = new ObjectOutputStream(new FileOutputStream(pathname));
            objectOutputStream.writeObject(object);
            objectOutputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public Object deserialize(String pathname) {
        // Leer un objeto serializado
        try {
            ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream(pathname));
            Object data = objectInputStream.readObject();
            objectInputStream.close();
            return data;
        } catch (FileNotFoundException e){
            System.out.println("No existen archivos binarios guardados. ");
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Log de Errores

Se debe tener un log de acciones que registre los cambios en los datos.
El Logse creó con el siguiente método:

```java
public static void addToEndFile(String pathname, String data) {
    FileWriter flwriter = null;
    try {
        flwriter = new FileWriter(pathname, true); // True indica que se va a agregar datos al final
        BufferedWriter bfwriter = new BufferedWriter(flwriter);
        // Escribe los datos en el archivo
        bfwriter.write(data);
        bfwriter.close();

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (flwriter != null) {
            try {
                flwriter.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

En el log se escriben los inicios de sesión exitosos y fallidos así como las eliminaciones de datos que haga el usuario en el menú principal.

```
if (username.equals(user.getUsername()) && pass.equals(user.getPassword())) {

    String log = fechaHoraActuales
            + "\t\t" + user.getUsername() + ": Inicio de sesión exitoso.\r\n";
    Log.addToEndFile("log.log", log);
    menu.menu();

} else if (username.equals(user.getUsername()) && !pass.equals(user.getPassword())) {
    System.out.println("Contraseña incorrecta, intente de nuevo. \r\n");
    String log = fechaHoraActuales
            + "\t\t" + user.getUsername() + ": Inicio de sesión fallido.\r\n";
    Log.addToEndFile("log.log", log);
}
usuario++;
```