

Proyecto Global

Programación

Burgos Lucas

Chagnaud Luciano

Filippini Juan Cruz

Navarro Pilar

Videla Franco



Enunciado elegido

OBJETOS Ejercicio 2

Consultora

Página 1 de 6

Enunciados de Proyectos para el examen Global

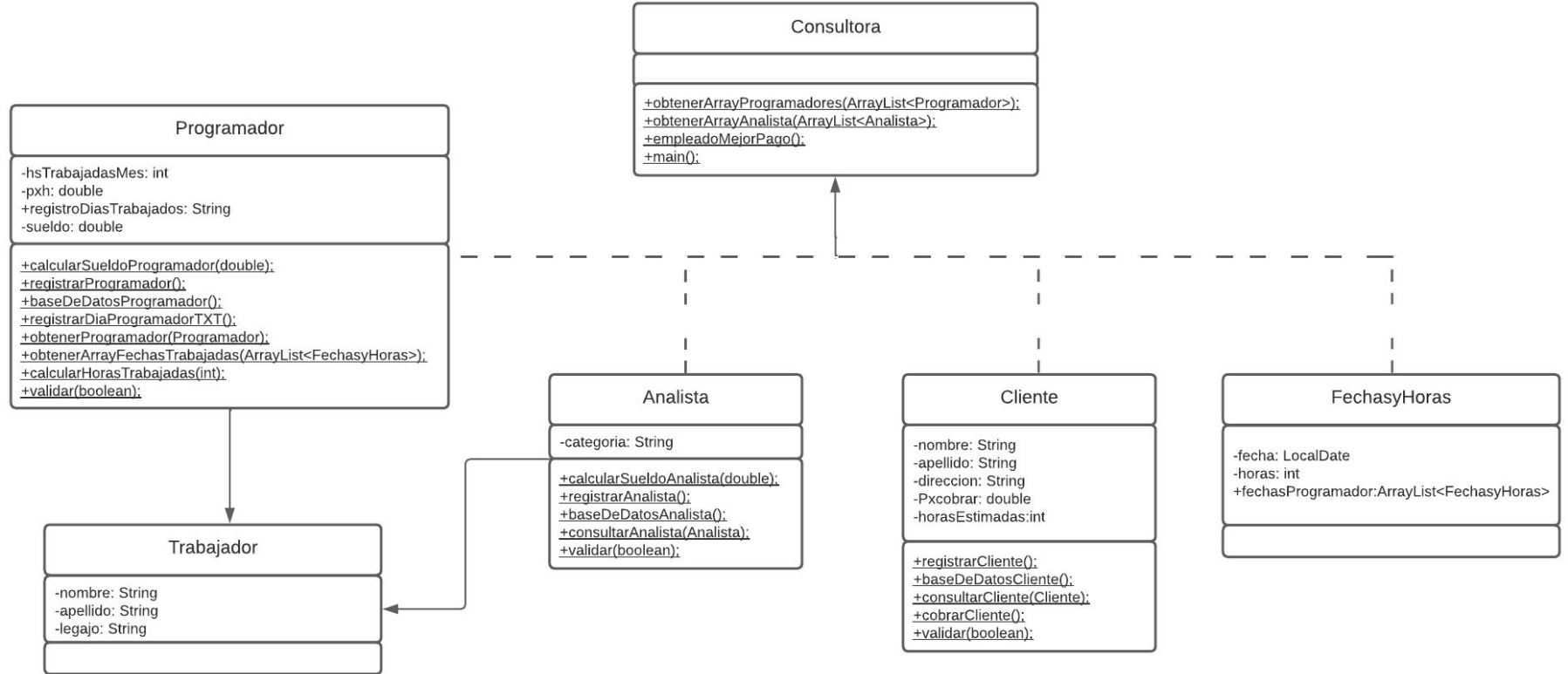
Una consultora desea realizar un sistema que le permita realizar la liquidación de haberes de sus empleados y de cobros a sus clientes.

- La consultora cuenta con un plantel de programadores y analistas, que desarrollan tareas a distintos clientes que contratan los servicios de la consultora.
- Los analistas cobran un sueldo fijo mensual que depende de su categoría y los programadores cobran por cada hora que trabajan un monto que acuerda cada uno con la consultora.
- Se registran las horas que los programadores trabajan por día a cada cliente de la consultora.
- De cada cliente que contrata a la consultora se registra el nombre, la dirección y el precio hora que fue acordado.

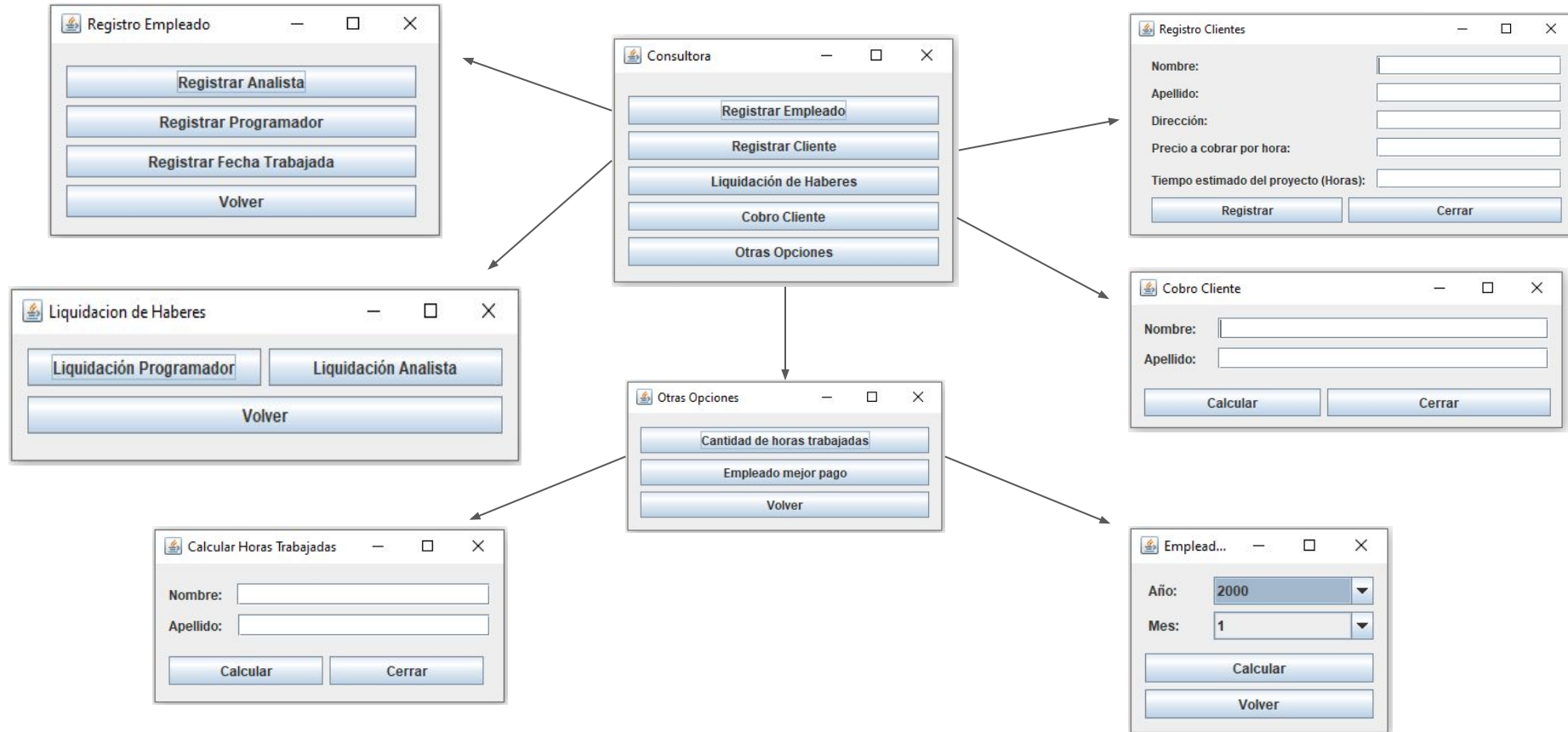
Realizar los métodos:

1. **liquidacionDesde: dia hasta: otroDia** Devuelve el valor que debe cobrar en total la consultora por las horas que trabajaron sus programadores en el periodo indicado.
2. **horasTrabajadasEn** Devuelve la cantidad de horas trabajadas por un programador desde su ingreso a la consultora
3. **empleadoMejorPago: mes y: año** Devuelve el empleado de la empresa que mejor sueldo cobra en el mes y año indicado.

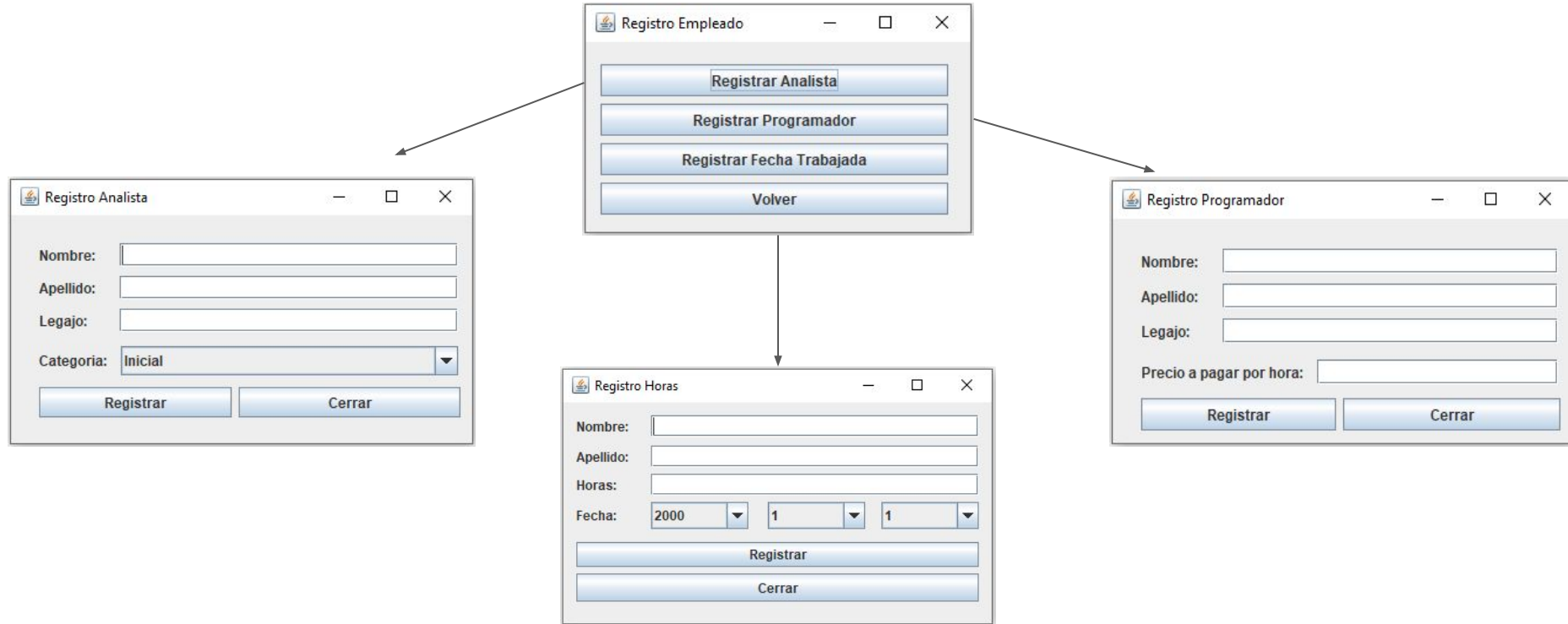
UML Consultora



Pantallas de la vista



Pantallas de la vista



Pantallas de la vista

```
graph TD; A[Liquidacion de Haberes] --> B[Calcular Sueldo Programador]; A --> C[Calcular Sueldo Analista];
```

Liquidacion de Haberes

Liquidación Programador

Liquidación Analista

Volver

Calcular Sueldo Programador

Nombre:

Apellido:

Calcular desde fecha: 2000 1 1

Calcular hasta fecha: 2000 1 1

Calcular

Cerrar

Calcular Sueldo Analista

Nombre:

Apellido:

Calcular

Cerrar

Resultados

Registro Analista

Nombre:

Apellido:

Legajo:

Categoría:

Message

Analista registrado exitosamente

Registro Programador

Nombre:

Apellido:

Legajo:

Precio a pagar por hora:

Message

Programador registrado exitosamente

Registro Horas

Nombre:

Apellido:

Horas:

Fecha:

Message

Registro existoso

Registro Clientes

Nombre:

Apellido:

Dirección:

Precio a cobrar por hora:

Tiempo estimado del proyecto (Horas):

Message

Cliente registrado exitosamente

Calcular Sueldo Programador

Nombre:

Apellido:

Calcular desde fecha:

Calcular hasta fecha:

Message

Sueldo a liquidar de: LUCIANO CHAGNAUD
\$16800.0

Calcular Sueldo Analista

Nombre:

Apellido:

Message

Sueldo a liquidar: \$120000.0


Resultados

Cobro Cliente

Nombre:

Apellido:

Message


 El presupuesto estimado es de: \$ 178500.0

Calcular Horas Trabajadas

Nombre:

Apellido:

Message


 Horas TOTALES trabajadas de: LUCIANO CHAGNAUD
34 HORAS

Empleador Mejor Pago

Año:

Mes:

Message

 El programador mejor pago es: LAUTARO MARTINEZ
Con un sueldo de: \$14000,00

El analista mejor pago es: PEDRO PASCAL
con un sueldo de: \$500000,00

Base de Datos (Clientes)

clientes.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

Nombre: ILARDO; Apellido: LOPEZ; Dirección: BORGES 333; Precio por cobrar: \$1800.0; Horas estimadas proyecto: 150;

Nombre: MARIO; Apellido: PERGOLINI; Dirección: AVENIDA SIEMPRE VIVA 743; Precio por cobrar: \$1500.0; Horas estimadas proyecto: 70;

Nombre: JESUS; Apellido: DATOLO; Dirección: MORON 456; Precio por cobrar: \$1600.0; Horas estimadas proyecto: 100;

Nombre: LAURA; Apellido: FRANCESCOI; Dirección: DIEGO MILITO 332; Precio por cobrar: \$2200.0; Horas estimadas proyecto: 90;

Nombre: PEDRO; Apellido: ARANDA; Dirección: SAN MARTIN SUR 1450; Precio por cobrar: \$1200.0; Horas estimadas proyecto: 65;

Nombre: LUCRECIA; Apellido: BENITEZ; Dirección: CASTELLI 989; Precio por cobrar: \$2300.0; Horas estimadas proyecto: 132;

Base de Datos (Analistas)

analistas.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

Nombre: LEANDRO; Apellido: CRUZ; Legajo: 001; Categoria: Inicial;

Nombre: JUAN; Apellido: PEREZ; Legajo: 002; Categoria: Superior;

Nombre: PEDRO; Apellido: PASCAL; Legajo: 003; Categoria: Senior;

Nombre: PABLO; Apellido: MINUZZI; Legajo: 004; Categoria: Intermedio;

Nombre: FRANCO; Apellido: PEREYRA; Legajo: 005; Categoria: Inicial;

Nombre: LUCIANA; Apellido: ALVAREZ; Legajo: 006; Categoria: Superior;

Nombre: CARLA; Apellido: DIAZ; Legajo: 007; Categoria: Inicial;

Nombre: PEDRO; Apellido: GONZALEZ; Legajo: 008; Categoria: Intermedio;

Nombre: LUCIA; Apellido: BENITEZ; Legajo: 009; Categoria: Superior;

Nombre: LORENZO; Apellido: SALVATIERRA; Legajo: 010; Categoria: Inicial;

Base de Datos (Programadores)

programadores.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

Nombre: LUCIANO; Apellido: CHAGNAUD; Legajo: 001; Sueldo por hora: 1200.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES
\REGISTRO PERSONAL\LUCIANOCHAGNAUDPersonal.txt

Nombre: PILAR; Apellido: NAVARRO; Legajo: 002; Sueldo por hora: 900.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES
\REGISTRO PERSONAL\PILARNAVARROPersonal.txt

Nombre: LUCAS; Apellido: MONSANTO; Legajo: 003; Sueldo por hora: 500.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES
\REGISTRO PERSONAL\LUCASMONSANTOPersonal.txt

Nombre: CARLA; Apellido: DIAZ; Legajo: 004; Sueldo por hora: 1500.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES\REGISTRO
PERSONAL\CARLADIAZPersonal.txt

Nombre: IGNACIO; Apellido: AMEAL; Legajo: 005; Sueldo por hora: 750.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES
\REGISTRO PERSONAL\IGNACIOAMEALPersonal.txt

Nombre: TIZIANO; Apellido: HIDALGO; Legajo: 006; Sueldo por hora: 980.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES
\REGISTRO PERSONAL\TIZIANOHIDALGOPersonal.txt

Nombre: LAURA; Apellido: LOPEZ; Legajo: 008; Sueldo por hora: 1300.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES
\REGISTRO PERSONAL\LAURALOPEZPersonal.txt

Nombre: JOAQUIN; Apellido: LIMA; Legajo: 007; Sueldo por hora: 350.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES
\REGISTRO PERSONAL\JOAQUINLIMAPersonal.txt

Nombre: LAUTARO; Apellido: MARTINEZ; Legajo: 009; Sueldo por hora: 1750.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES
\REGISTRO PERSONAL\LAUTAROMARTINEZPersonal.txt

Nombre: FRANCO; Apellido: VIDELA; Legajo: 010; Sueldo por hora: 1250.0; Registro Personal: BASE DE DATOS\EMPLEADOS\PROGRAMADORES
\REGISTRO PERSONAL\FRANCOVIDELAPersonal.txt

Base de Datos (Registro Personal)



LUCIANOCHAGNAUDPersonal.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

Fecha: 2022-03-03; horas: 20

Fecha: 2021-05-08; horas: 5

Fecha: 2021-12-10; horas: 9

Código Completo Trabajador

```
package consultora;

public class Trabajador {
    private String nombre;
    private String apellido;
    private String legajo;

    public Trabajador() {
    }

    public Trabajador(String nombre, String apellido, String legajo) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.legajo = legajo;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }
}
```

Código completo Analista

```
package consultora;

import java.io.*;
import javax.swing.*;

public class Analista extends Trabajador {

    private String categoria;

    public Analista() {
    }

    public Analista(String nombre, String apellido, String legajo) {
        super(nombre, apellido, legajo);
    }

    public Analista(String categoria, String nombre, String apellido, String legajo) {
        super(nombre, apellido, legajo);
        this.categoria = categoria;
    }

    public String getCategoria() {
        return categoria;
    }

    public void setCategoria(String categoria) {
        this.categoria = categoria;
    }
}
```


Código completo Analista

```
//Permite la liquidación de haberes de los analistas
public static double calcularSueldoAnalista(String nombre, String apellido) {
    //creamos la variable sueldo
    double sueldo = 0;

    //Extraemos de la base de datos el analista solicitado
    Analista analista = consultarAnalista(nombre, apellido);

    //Calculamos el sueldo en base a su categoria
    switch (analista.getCategoria()) {
        case "Inicial" ->
            sueldo = 70000;
        case "Intermedio" ->
            sueldo = 120000;
        case "Superior" ->
            sueldo = 200000;
        case "Senior" ->
            sueldo = 500000;
    }
    return sueldo;
}
```

Código completo Analista

```
//Registra a un Analista en la base de datos
public static void registrarAnalista(String nombre, String apellido, String legajo, String categoria) throws IOException {

    //Corroboramos que el analista no exista previamente en la base de datos
    boolean valido = validar(nombre, apellido);

    if (valido) {
        Analista analista = new Analista(categoria, nombre, apellido, legajo);

        //Guardar los analistas en una base de datos (txt)
        baseDeDatosAnalista(ana: analista);
    }
}
```


Código completo Analista

```
//Base de datos de los analistas
public static void baseDeDatosAnalista(Analista ana) throws FileNotFoundException, IOException {
    try {
        // Crear un FileWriter para escribir en el archivo de texto
        FileWriter fileWriter = new FileWriter("BASE DE DATOS\\EMPLEADOS\\ANALISTAS\\analistas.txt", append: true);

        // Crear un BufferedWriter para escribir en el FileWriter
        BufferedWriter writer = new BufferedWriter(out: fileWriter);

        // Obtener los atributos del analista
        String nombre = ana.getNombre();
        String apellido = ana.getApellido();
        String legajo = ana.getLegajo();
        String categoria = ana.getCategoria();

        // Escribir los atributos en el archivo de texto
        writer.write("Nombre: " + nombre + "; ");
        writer.write("Apellido: " + apellido + "; ");
        writer.write("Legajo: " + legajo + "; ");
        writer.write("Categoria: " + categoria + "; ");
        writer.newLine();
        writer.newLine();

        // Cerrar el BufferedWriter
        writer.close();

        JOptionPane.showMessageDialog(parentComponent: null, message: "Analista registrado exitosamente");
    } catch (IOException e) {
        System.out.println("Error al registrar el analista: " + e.getMessage());
    }
}
```

Código completo Analista

```
//Buscar en la Base de Datos un Analista solicitado por el Usuario
public static Analista consultarAnalista(String nombre, String apellido) {
    try {
        // Crear un FileReader para leer el archivo de texto
        FileReader fileReader = new FileReader("BASE DE DATOS\\EMPLEADOS\\ANALISTAS\\analistas.txt");

        // Crear un BufferedReader para leer el FileReader
        BufferedReader reader = new BufferedReader(in:fileReader);

        String linea;
        //Bucle para recorrer el archivo
        while ((linea = reader.readLine()) != null) {
            //Verificamos si el nombre que ingresa el usuario coincide con el nombre en la base de datos
            if (linea.contains( ":" + nombre) & linea.contains( ":" + apellido)) {
                // Extraer los valores de los atributos
                //Lo que hace este metodo es separar el string en distintos substring, utilizando el delimitador ":"
                //Luego con el [] accedemos al valor y se lo asignamos a una variable.
                String[] atributos = linea.split(":" );
                String nombreAnalista = atributos[1];
                String apellidoAnalista = atributos[3];
                String legajo = atributos[5];
                String categoria = atributos[7];

                // Crear un nuevo objeto Analista con los atributos leídos
                Analista analista = new Analista(categoria, nombre: nombreAnalista, apellido: apellidoAnalista, legajo);

                reader.close();
                return analista;
            }
        }
    }
}
```

Código completo Analista

```
public static boolean validar(String nombre, String apellido) {  
    // Verificar si el nombre o apellido están vacíos o son nulos  
    if (nombre == null || apellido == null || nombre.equals(anObject: "") || apellido.equals(anObject: "")) {  
        JOptionPane.showMessageDialog(parentComponent: null, message: "El nombre o apellido están vacíos, ingrese uno válido");  
        return false; // Si la condición se cumple, se muestra un mensaje y se retorna false  
    }  
  
    Analista existe = consultarAnalista(nombre, apellido); // Consultar si existe un analista con el nombre y apellido dados  
    boolean esValido = false;  
  
    // Verificar si no se encontró un analista con el mismo nombre y apellido  
    if (existe == null) {  
        esValido = true; // Si no se encontró, se marca como válido  
    }  
  
    // Mostrar un mensaje si los datos ya están registrados en la base de datos  
    if (!esValido) {  
        JOptionPane.showMessageDialog(parentComponent: null, message: "Los datos ingresados ya están registrados en nuestra base de datos");  
    }  
  
    return esValido; // Retornar el valor de esValido (true si es válido, false si no)  
}
```

Código completo Cliente

```
package consultora;

import java.io.*;
import javax.swing.*;

public class Cliente {

    private String nombre;
    private String apellido;
    private String direccion;
    private double Pxcobrar;
    private int horasEstimadas;

    public Cliente() {
    }

    public Cliente(String nombre, String apellido, String direccion, double Pxcobrar, int horasEstimadas) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.direccion = direccion;
        this.Pxcobrar = Pxcobrar;
        this.horasEstimadas = horasEstimadas;
    }

    public int getHorasEstimadas() {
        return horasEstimadas;
    }

    public void setHorasEstimadas(int horasEstimadas) {
        this.horasEstimadas = horasEstimadas;
    }
}
```

Código completo Cliente

```
//Registra a un Cliente en la base de datos
public static void registrarCliente(String nombre, String apellido, String direccion, double pxh, int horasEstimadas) throws

    boolean valido = validar(nombre, apellido);

    if (valido) {
        //Instanciamos un nuevo objeto cliente con los datos obtenidos de la interfaz
        Cliente cliente = new Cliente(nombre, apellido, direccion, pxh, horasEstimadas);

        //Guardamos los atributos del objeto en una base de datos (TXT)
        baseDeDatosCliente(cliente);
    }
}
```

Código completo Cliente

```
//Base de datos de los clientes
public static void baseDeDatosCliente(Cliente cliente) throws IOException {

    try {
        FileWriter fileWriter = new FileWriter(fileName: "BASE DE DATOS\\CLIENTES\\clientes.txt", append: true);

        // Crear un BufferedWriter para escribir en el FileWriter
        BufferedWriter writer = new BufferedWriter(out: fileWriter);
        // Obtener los atributos del analista
        String nombre = cliente.getNombre();
        String apellido = cliente.getApellido();
        String direccion = cliente.getDireccion();
        double pxcobrar = cliente.getPxcobrar();
        int horasEstimadas = cliente.getHorasEstimadas();

        // Escribir los atributos en el archivo de texto
        writer.write("Nombre: " + nombre + "; ");
        writer.write("Apellido: " + apellido + "; ");
        writer.write("Dirección: " + direccion + "; ");
        writer.write("Precio por cobrar: $" + pxcobrar + "; ");
        writer.write("Horas estimadas proyecto: " + horasEstimadas + "; ");
        writer.newLine();
        writer.newLine();

        // Cerrar el BufferedWriter
        writer.close();

        JOptionPane.showMessageDialog(parentComponent: null, message: "Cliente registrado exitosamente");
    } catch (IOException e) {
        System.out.println("Error al registrar el cliente: " + e.getMessage());
    }
}
```


Código completo Cliente

```
114 //Buscar en la Base de Datos un Cliente solicitado por el Usuario
115 public static Cliente consultarCliente(String nombre, String apellido) {
116     try {
117
118         // Crear un FileReader para leer el archivo de texto
119         FileReader fileReader = new FileReader("BASE DE DATOS\\CLIENTES\\clientes.txt");
120
121         // Crear un BufferedReader para leer el FileReader
122         BufferedReader reader = new BufferedReader(in:fileReader);
123
124         String linea;
125         //Bucle para recorrer el archivo
126         while ((linea = reader.readLine()) != null) {
127             //Verificamos si el nombre que ingresa el usuario coincide con el nombre en la base de datos
128             if (linea.contains(:nombre) & linea.contains(:apellido)) {
129                 // Extraer los valores de los atributos
130                 //Lo que hace este metodo es separar el string en distintos substring, utilizando el delimitador ":"
131                 //Luego con el [] accedemos al valor y se lo asignamos a una variable.
132                 String[] atributos = linea.split(regex:" |; |$ ");
133                 String nombreCliente = atributos[1];
134                 String apellidoCliente = atributos[3];
135                 String direccion = atributos[5];
136                 String PxcobrarString = atributos[7].replaceAll(regex:"\\$", replacement:"");
137                 double Pxcobrar = Double.parseDouble(:PxcobrarString);
138                 int horasEstimadas = Integer.parseInt(atributos[9]);
139                 // Crear un nuevo objeto Cliente con los atributos leídos
140                 Cliente cliente = new Cliente(nombre:nombreCliente, apellido:apellidoCliente, direccion, Pxcobrar, horasEstimadas);
141                 reader.close();
142                 return cliente;
143             }
144         }
```

Código completo Cliente

```
//Permite calcular cuanto se le cobrara al cliente, con un presupuesto estimado
public static void cobrarCliente(String nombre, String apellido) {

    if (nombre.equals("") || apellido.equals("")) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "El nombre o apellido están vacíos, ingrese uno válido");
        return;
    }

    Cliente cliente = consultarCliente(nombre, apellido);

    if (cliente == null) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "El cliente ingresado no se encuentra en la base de datos.");
    } else {
        JOptionPane.showMessageDialog(parentComponent: null, "El presupuesto estimado es de: $ " + cliente.getHorasEstimadas() * cliente.ge

    }

}
```


Código completo Cliente

```
}  
//Validamos si el cliente ya existe  
public static boolean validar(String nombre, String apellido) {  
    if (nombre.equals("") || apellido.equals("")) {  
        JOptionPane.showMessageDialog(parentComponent: null, message: "El nombre o apellido están vacíos, ingrese uno válido");  
        return false;  
    }  
  
    Cliente existe = consultarCliente(nombre, apellido);  
    boolean esValido = false;  
  
    if (existe == null) {  
        esValido = true;  
    }  
  
    if (!esValido) {  
        JOptionPane.showMessageDialog(parentComponent: null, message: "Los datos ingresados ya están registrados en nuestra base de datos");  
    }  
  
    return esValido;  
}
```

Código completo Programador

```
package consultora;

import java.io.*;
import javax.swing.*;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class Programador extends Trabajador {

    private int hsTrabajadasMes;
    private double pxh;
    String registroDiasTrabajados;
    private double sueldo;

    public Programador() {
    }

    public Programador(String nombre, String apellido, String legajo) {
        super(nombre, apellido, legajo);
    }

    public Programador(double pxh, String nombre, String apellido, String legajo) {
        super(nombre, apellido, legajo);
        this.pxh = pxh;
    }

    public Programador(double pxh, String registroDiasTrabajados, String nombre, String apellido, String legajo) {
        super(nombre, apellido, legajo);
        this.pxh = pxh;
        this.registroDiasTrabajados = registroDiasTrabajados;
    }
}
```

Código completo Programador

```
75 //Permite la liquidación de haberes de los programadores
76 public static double calcularSueldoProgramador(String nombre, String apellido, LocalDate fechaDesde, LocalDate fechaHasta) throws :
77     //creamos la variable sueldo
78     double sueldo = 0;
79
80     //Extraemos al programador solicitado de la base de datos
81     Programador programador = obtenerProgramador(nombre, apellido);
82
83     //Obtenemos un ArrayList con las fechas trabajadas por el programador
84     ArrayList<FechasYHoras> fechasProgramador = obtenerArrayFechasTrabajadas(nombre, apellido);
85
86     //Calculamos las horas trabajadas en el año y mes solicitado
87     int horas = calcularHorasTrabajadas(fechasProgramador, fechaDesde, fechaHasta);
88
89     //Calculamos el sueldo que corresponderia en base a las horas trabajadas y cuanto se le paga por hora
90     sueldo = programador.getPxh() * horas;
91     return sueldo;
92
93 }
```

Código completo Programador

```
//Registra a un Programador en la base de datos
public static void registrarProgramador(String nombre, String apellido, String legajo, double pxh) throws IOException {

    //Validamos si el programador existe en la base de datos previamente
    boolean valido = validar(nombre, apellido, precio:pxh);

    if (valido) {

        Programador programador = new Programador(pxh, nombre, apellido, legajo);

        // Guardamos los datos en la base de datos
        baseDeDatosProgramador( prog: programador);

    }
}
```

Código completo Programador

```
//Base de datos de los programadores
public static void baseDeDatosProgramador(Programador prog) throws IOException, FileNotFoundException {
    try {
        String file = "BASE DE DATOS\\EMPLEADOS\\PROGRAMADORES\\";
        String nombreRegistro = prog.getNombre() + prog.getApellido();
        // Crear un FileWriter para escribir en el archivo de texto
        FileWriter fileWriter = new FileWriter(file + "programadores.txt", append: true);

        // Crear un BufferedWriter para escribir en el FileWriter
        BufferedWriter writer = new BufferedWriter(out: fileWriter);

        // Escribir los atributos en el archivo de texto
        writer.write("Nombre: " + prog.getNombre() + "; ");
        writer.write("Apellido: " + prog.getApellido() + "; ");
        writer.write("Legajo: " + prog.getLegajo() + "; ");
        writer.write("Sueldo por hora: " + prog.getPxh() + "; ");
        writer.write("Registro Personal: " + file + "REGISTRO PERSONAL\\" + nombreRegistro + "Personal.txt");
        writer.newLine();
        writer.newLine();

        // Cerrar el BufferedWriter
        writer.close();

        JOptionPane.showMessageDialog(parentComponent: null, message: "Programador registrado exitosamente");
    } catch (IOException e) {
        JOptionPane.showMessageDialog(parentComponent: null, "Error al registrar al Programador " + e.getMessage());
    }
}
```

Código completo Programador

```
//Registramos en un txt las fechas y horas trabajadas
public static void registrarDiaProgramadorTXT(String nombre, String apellido, LocalDate fechaLocalDate, String horas) throws IOException {
    if ("".equals( anObject:nombre) || "".equals( anObject:apellido) || "".equals( anObject:horas) || fechaLocalDate.isEqual( other:LocalDate.of
        JOptionPane.showMessageDialog( parentComponent:null, message:"Debe completar correctamente todos los campos");
        return;
    }

    Programador p = obtenerProgramador(nombre, apellido);

    if (p != null) {
        try {
            String nombreRegistro = nombre + apellido;
            String fileName = "BASE DE DATOS\\EMPLEADOS\\PROGRAMADORES\\REGISTRO PERSONAL\\" + nombreRegistro + "Personal.txt";

            FileWriter filewriter = new FileWriter(fileName, append:true);
            BufferedWriter writer = new BufferedWriter( out:filewriter);

            String fecha = fechaLocalDate.toString();
            writer.write("Fecha: " + fecha + "; ");
            writer.write("horas: " + horas);
            writer.newLine();
            writer.newLine();

            // Cerrar el BufferedWriter
            writer.close();

            JOptionPane.showMessageDialog( parentComponent:null, message:"Registro existoso");
        } catch (IOException e) {
            JOptionPane.showMessageDialog( parentComponent:null, "Error en el registro" + e.getMessage());
        }
    }
}
```


Código completo Programador

```
//Buscar en la Base de Datos un Programador solicitado por el Usuario
public static Programador obtenerProgramador(String nombre, String apellido) {
    try {

        // Crear un FileReader para leer el archivo de texto
        FileReader fileReader = new FileReader("BASE DE DATOS\\EMPLEADOS\\PROGRAMADORES\\programadores.txt");

        // Crear un BufferedReader para leer el FileReader
        BufferedReader reader = new BufferedReader(in:fileReader);

        String linea;
        //Bucle para recorrer el archivo
        while ((linea = reader.readLine()) != null) {
            //Verificamos si el nombre que ingresa el usuario coincide con el nombre en la base de datos
            if (linea.contains(s.nombre) & linea.contains(s.apellido)) {
                // Extraer los valores de los atributos
                //Lo que hace este metodo es separar el string en distintos substring, utilizando el delimitador ":"
                //Luego con el [] accedemos al valor y se lo asignamos a una variable.
                String[] atributos = linea.split(regex: ": |");
                String nombreProgramador = atributos[1];
                String apellidoProgramador = atributos[3];
                String legajo = atributos[5];
                double pxx = Double.parseDouble(atributos[7]);
                String registro = atributos[9];

                // Crear un nuevo objeto Programador con los atributos leídos
                Programador programador = new Programador(pxx, registroDiasTrabajados:registro, nombre:nombreProgramador, apellido:apellido

                reader.close();
                return programador;
            }
        }
    }
}
```

Código completo Programador

/Busca en la base de datos PROPIA de cada programador y retorna un ArrayList ordenado con las fechas y horas trabajadas por cada programador

```
public static ArrayList<FechasYHoras> obtenerArrayFechasTrabajadas(String nombre, String apellido) throws IOException {
    try {

        String nombreRegistro = nombre + apellido;
        String fileName = "BASE DE DATOS\\EMPLEADOS\\PROGRAMADORES\\REGISTRO PERSONAL\\" + nombreRegistro + "Personal.txt";
        FileReader fileReader = new FileReader(fileName);
        BufferedReader reader = new BufferedReader(in:fileReader);

        String linea;
        ArrayList<FechasYHoras> fechasProgramador = new ArrayList();
        // Bucle para recorrer el archivo
        while ((linea = reader.readLine()) != null) {
            if (linea.contains(":Fecha")) {
                //Guardamos TODAS las fechas en un ArrayList
                String[] lineaArreglo = linea.split("regex:": "|");
                LocalDate fecha = LocalDate.parse(lineaArreglo[1]);
                int horas = Integer.parseInt(lineaArreglo[3]);
                FechasYHoras fechas = new FechasYHoras(fecha, horas);
                fechasProgramador.add(fechas);
            }
        }

        //Ordenamos el ArrayList
        Collections.sort(list:fechasProgramador, c:Comparator.comparing(FechasYHoras::getFecha));
        reader.close();
        return fechasProgramador;
    } catch (IOException e) {
        JOptionPane.showMessageDialog(parentComponent:null, message:e.getMessage());
        return null;
    }
}
```


Código completo Programador

```
54 //Calculamos las horas trabajadas del programador en base a un periodo de tiempo
55 public static int calcularHorasTrabajadas(ArrayList<FechasyHoras> fechasProgramador, LocalDate fechaDesde, LocalDate fechaHasta)
56 {
57     int horas = 0;
58     //Calculamos las horas trabajadas en base a la fecha especifica que el usuario solicita
59     if (fechaDesde == null & fechaHasta == null) {
60         for (FechasyHoras fecha : fechasProgramador) {
61             horas += fecha.getHoras();
62         }
63     } else {
64         //Calculamos las horas trabajadas desde que ingresó a la consultora
65         for (FechasyHoras fecha : fechasProgramador) {
66             if (fecha.getFecha().compareTo(others: fechaDesde) >= 0 & fecha.getFecha().compareTo(others: fechaHasta) <= 0) {
67                 horas += fecha.getHoras();
68             }
69         }
70     }
71
72     return horas;
73 }
```

Código completo Programador

```
public static boolean validar(String nombre, String apellido, double precio) {  
  
    // Verificar si el nombre o apellido están vacíos o son nulos  
    if (nombre == null || apellido == null || nombre.equals("") || apellido.equals("")) {  
        JOptionPane.showMessageDialog(parentComponent: null, message: "El nombre o apellido están vacíos, ingrese uno válido");  
        return false; // Si la condición se cumple, se muestra un mensaje y se retorna false  
    }  
  
    Programador existe = obtenerProgramador(nombre, apellido); // Consultar si existe un programador con el nombre y apellido dados  
    boolean esValido = false;  
  
    // Verificar si no se encontró un programador con el mismo nombre y apellido  
    if (existe == null) {  
        esValido = true; // Si no se encontró, se marca como válido  
    }  
  
    // Mostrar un mensaje si los datos ya están registrados en la base de datos  
    if (!esValido) {  
        JOptionPane.showMessageDialog(parentComponent: null, message: "Los datos ingresados ya están registrados en nuestra base de datos.");  
    }  
  
    return esValido; // Retornar el valor de esValido (true si es válido, false si no)  
}
```

Código completo FechasyHoras

```
1 package consultora;
2
3 import java.time.LocalDate;
4 import java.util.ArrayList;
5
6 public class FechasyHoras {
7     private LocalDate fecha;
8     private int horas;
9     public ArrayList<FechasyHoras> fechasProgramador;
10
11     public FechasyHoras() {
12     }
13
14     public FechasyHoras(LocalDate fecha, int horas) {
15         this.fecha = fecha;
16         this.horas = horas;
17     }
18
19     public LocalDate getFecha() {
20         return fecha;
21     }
22
23     public void setFecha(LocalDate fecha) {
24         this.fecha = fecha;
25     }
26
27     public int getHoras() {
28         return horas;
29     }
30
31     public void setHoras(int horas) {
32         this.horas = horas;
33     }
34 }
```

Código completo Consultora/Main

```
//Obtenemos un array de Programadores
public static ArrayList<Programador> obtenerArrayProgramadores() throws IOException {
    try {
        ArrayList<Programador> programadores = new ArrayList();
        // Crear un FileReader para leer el archivo de texto
        FileReader fileReader = new FileReader("BASE DE DATOS\\EMPLEADOS\\PROGRAMADORES\\programadores.txt");

        // Crear un BufferedReader para leer el FileReader
        BufferedReader reader = new BufferedReader(in:fileReader);
        String linea;
        //Bucle para recorrer el archivo
        while ((linea = reader.readLine()) != null) {
            //Verificamos si el nombre que ingresa el usuario coincide con el nombre en la base de datos
            // Extraer los valores de los atributos
            if (linea.contains(":Nombre")) {
                String[] atributos = linea.split("regex:": "|");
                String nombreProgramador = atributos[1];
                String apellidoProgramador = atributos[3];
                String legajo = atributos[5];
                double pxx = Double.parseDouble(atributos[7]);
                String registro = atributos[9];

                // Crear un nuevo objeto Programador con los atributos leídos
                Programador programador = new Programador(pxx, registroDiasTrabajados:registro, nombre:nombreProgramador, apellido:apellidoProgramador);

                //Agregamos al programador al ArrayList
                programadores.add(e:programador);
            }
        }
        reader.close();
        return programadores;
    }
}
```

Código completo Consultora/Main

```
public static ArrayList<Analista> obtenerArrayAnalista() {
    try {
        ArrayList<Analista> analistas = new ArrayList();
        // Crear un FileReader para leer el archivo de texto
        FileReader fileReader = new FileReader("BASE DE DATOS\\EMPLEADOS\\ANALISTAS\\analistas.txt");

        // Crear un BufferedReader para leer el FileReader
        BufferedReader reader = new BufferedReader(in: fileReader);

        String linea;
        //Bucle para recorrer el archivo
        while ((linea = reader.readLine()) != null) {
            //Verificamos si el nombre que ingresa el usuario coincide con el nombre en la base de datos
            // Extraer los valores de los atributos
            if (linea.contains(":Nombre")) {
                String[] atributos = linea.split(regex: ": |; ");
                String nombre = atributos[1];
                String apellido = atributos[3];
                String legajo = atributos[5];
                String categoria = atributos[7];

                // Crear un nuevo objeto Analista con los atributos leídos
                Analista analista = new Analista(categoria, nombre, apellido, legajo);

                //Agregamos al analista al ArrayList
                analistas.add(e: analista);
            }
        }
    }
}
```

Código completo Consultora/Main

```
1 public static void empleadoMejorPago(ArrayList<Programador> programadores, ArrayList<Analista> analistas, LocalDate fechaDesde, LocalTime horaDesde, LocalDate fechaHasta, LocalTime horaHasta) {
    double sueldoMayorProg = 0;
    Programador progMejorPagoMes = new Programador();
    for (Programador programador : programadores) {
        double sueldo = Programador.calcularSueldoProgramador(nombre: programador.getNombre(), apellido: programador.getApellido(), fechaDesde: fechaDesde, horaDesde: horaDesde, fechaHasta: fechaHasta, horaHasta: horaHasta);
        if (sueldo > sueldoMayorProg) {
            sueldoMayorProg = sueldo;
            progMejorPagoMes.setNombre(nombre: programador.getNombre());
            progMejorPagoMes.setApellido(apellido: programador.getApellido());
        }
    }
    double sueldoMayorAna = 0;
    Analista anaMejorPagoMes = new Analista();
    for (Analista analista : analistas) {
        double sueldo = Analista.calcularSueldoAnalista(nombre: analista.getNombre(), apellido: analista.getApellido(), fechaDesde: fechaDesde, horaDesde: horaDesde, fechaHasta: fechaHasta, horaHasta: horaHasta);
        if (sueldo > sueldoMayorAna) {
            sueldoMayorAna = sueldo;
            anaMejorPagoMes.setNombre(nombre: analista.getNombre());
            anaMejorPagoMes.setApellido(apellido: analista.getApellido());
        }
    }
    String mensaje = ""
    El programador mejor pago es: %s %s
    Con un sueldo de: $%.2f

    El analista mejor pago es: %s %s
    con un sueldo de: $%.2f
    "";
```


Código completo Consultora/Main

```
String nombreProg = progMejorPagoMes.getNombre();
String apellidoProg = progMejorPagoMes.getApellido();
String nombreAna = anaMejorPagoMes.getNombre();
String apellidoAna = anaMejorPagoMes.getApellido();
String mensajeCompleto = String.format(format:mensaje, args: nombreProg, args: apellidoProg, args: sueldoMayorProg, args: nombreAna, as
JOptionPane.showMessageDialog(parentComponent: null, message: mensajeCompleto);
}

public static void main(String[] args) throws IOException {

    InterfazPrincipal frame = new InterfazPrincipal();
    frame.setVisible(b: true);

}
```