BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

INGENIERIA EN CIENCIAS DE LA COMPUTACIÓN

BASES DE DATOS PARA INGENIERIA

PROYECTO FINAL PARTE I:

IMPLEMENTACIÓN DE UN DESCRIPTOR DE ARCHIVOS PARA LA LECTURA DE UNA TABLA OBTENIDA DE UNA BASE DE DATOS

INTEGRANTES:

Báez González José	201657079
Bautista Otero Alexandra	201640295
Coria Ríos Marco Antonio	201734576
Hernández Ramos Ángel	201653224
Torres Pérez Daniel	201733929

DOCENTE:

Alma Delia Ambrosio Vázquez

IMPLEMENTACIÓN DE UN DESCRIPTOR DE ARCHIVOS PARA LA LECTURA DE UNA TABLA OBTENIDA DE UNA BASE DE DATOS.

ÍNDICE

INTRODUCCIÓN

En la primera parte de la entrega del proyecto final nos enfocamos en la creación de un modelo estático que nos ayude a aplicar el funcionamiento analizado en clase de un descriptor de archivos para un caso específico; en este caso la consulta de un **Between-And** el cual hemos programado en Java y se encarga de leer la base de un archivo de texto (en formato TXT), almacenarla en una estructura para posteriormente aplicar los procedimientos que lleva a cabo el Manejador.

OBJETIVOS

Llevar a la practica la teoría analizada en la materia acerca del funcionamiento de un manejador de archivos de tal modo que sea observable cada paso del procedimiento que lleva a cabo para mostrar resultados, para que concuerden con la consulta que se ha seleccionado a "mano" mediante el uso de Excel, entonces plasmarlo en un programa con interfaz gráfica que se encargue de seguir estos pasos, imprimiendo en pantalla cada uno de ellos.

MARCO TEÓRICO

Descriptor de Archivos: Es un bloque de control que el sistema manejador de bases de datos (DBMS) utiliza para poder interpretar y almacenar los datos de una tabla. En éste se definen las características de los atributos de la tabla, las cuales son: **nombre del archivo**, **extensión** (cuántos espacios ocupa), y puede también especificar si se permiten datos nulos o no, entre otros elementos, dependiendo del DBMS.

Sistema Manejador De Bases De Datos: Es un programa que ofrece las herramientas necesarias para el manejo de una base de datos, se utilizan para diferentes **consultas** las cuales arrojan distintos resultados dependiendo de las especificaciones requeridas.

Operador Between: La función del operador Between es seleccionar valores dentro de un rango, dichos valores pueden ser números, texto o fechas; el operador mencionado se ocupa con la cláusula WHERE.

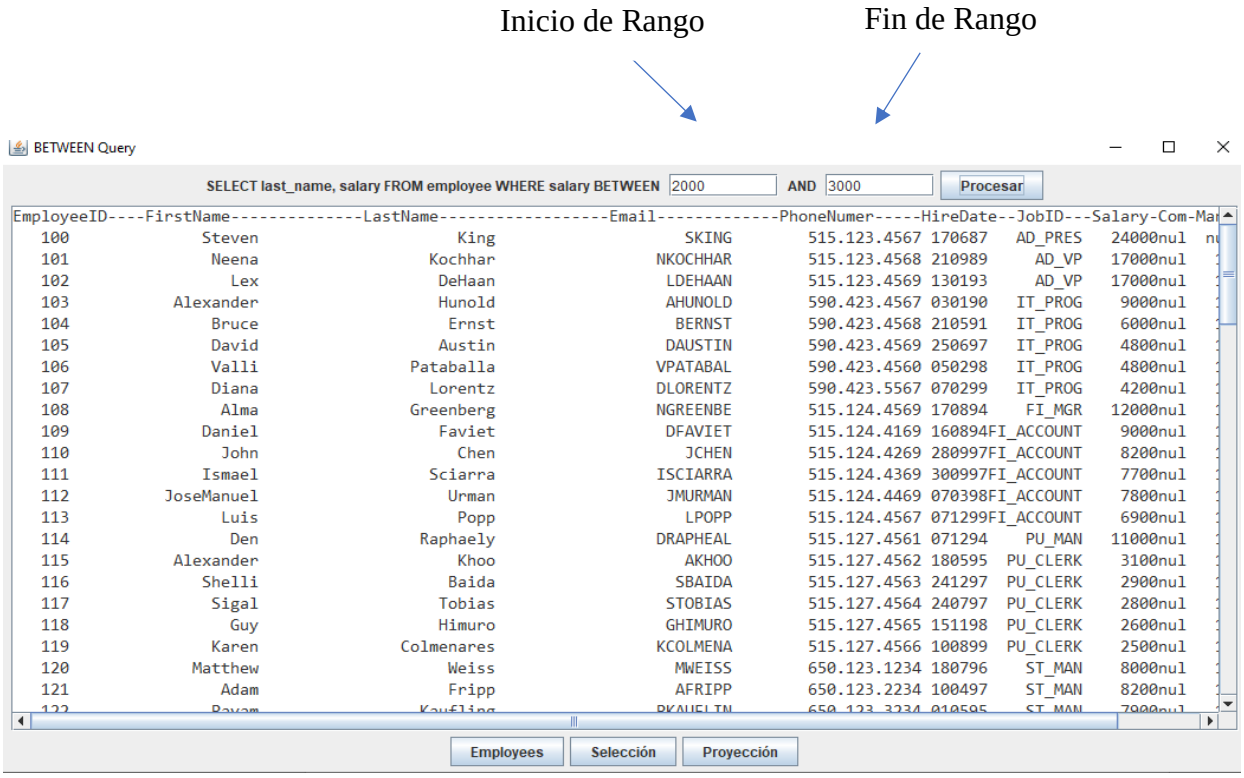
Sintaxis:

```
SELECT columna
FROM tabla WHERE columna
BETWEEN valor1 AND valor2
```

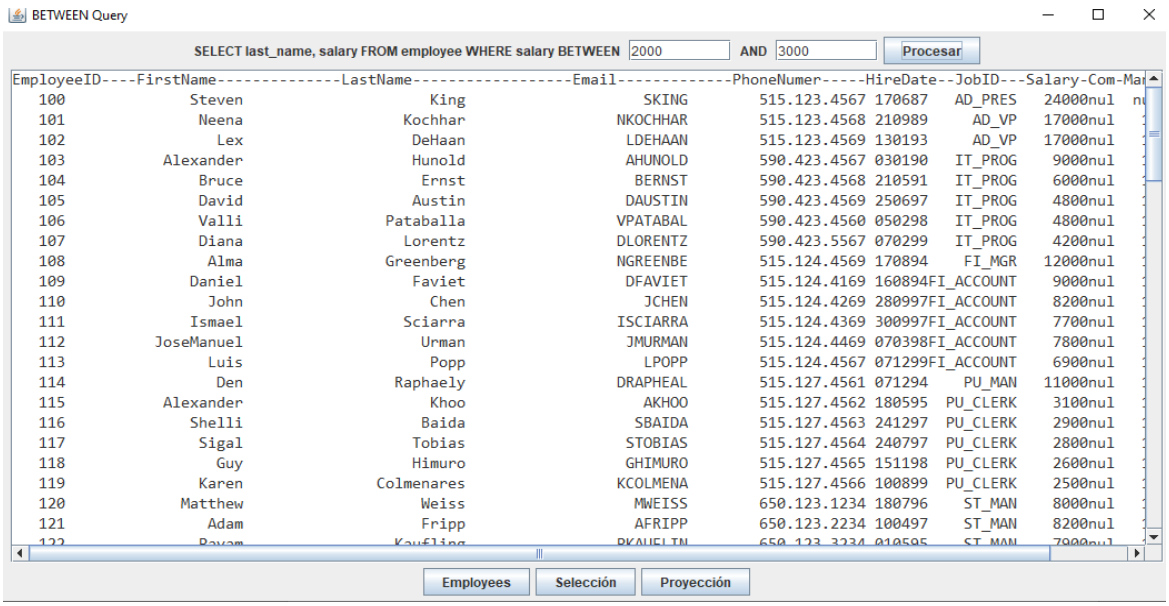
DESCRIPCIÓN DE INTERFAZ

La ventana principal proporciona dos cajas de texto que permiten al usuario ingresar el inicio y el fin del rango a utilizar; de igual manera contiene el botón “Procesar” el cual será el encargado de avisar al programa que es tiempo de empezar.

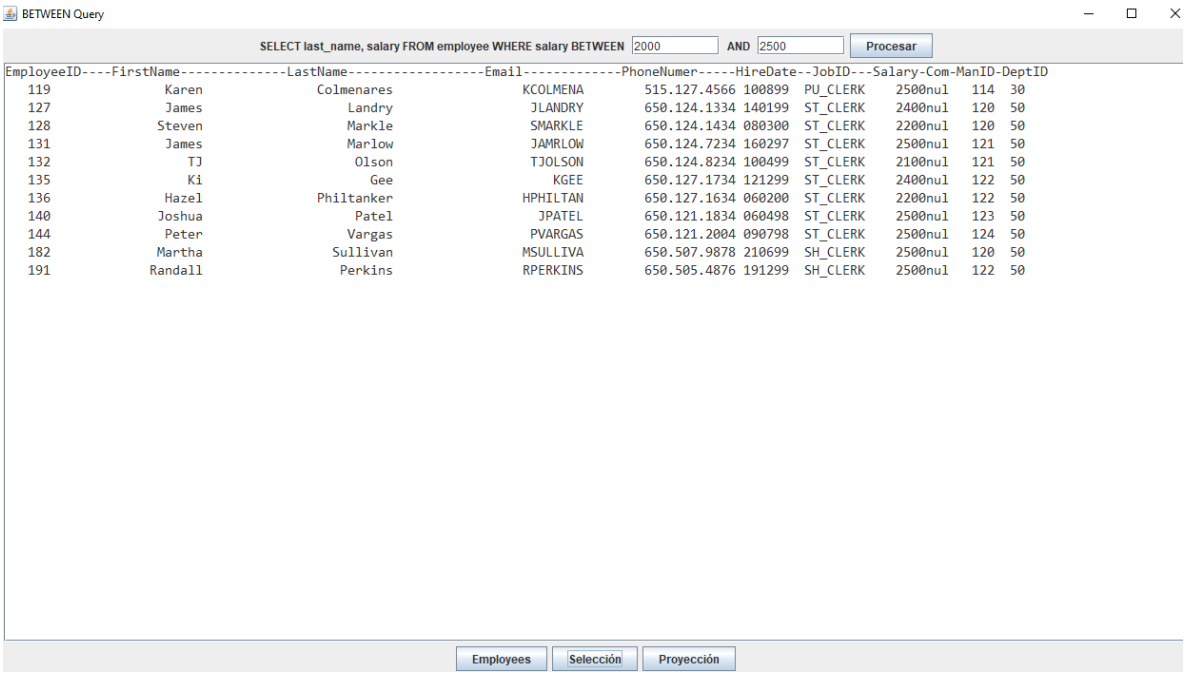
En la parte inferior central se hayan otros tres botones, los cuales mostraran las diferentes etapas del proceso que se lleva a cabo para obtener el resultado final.



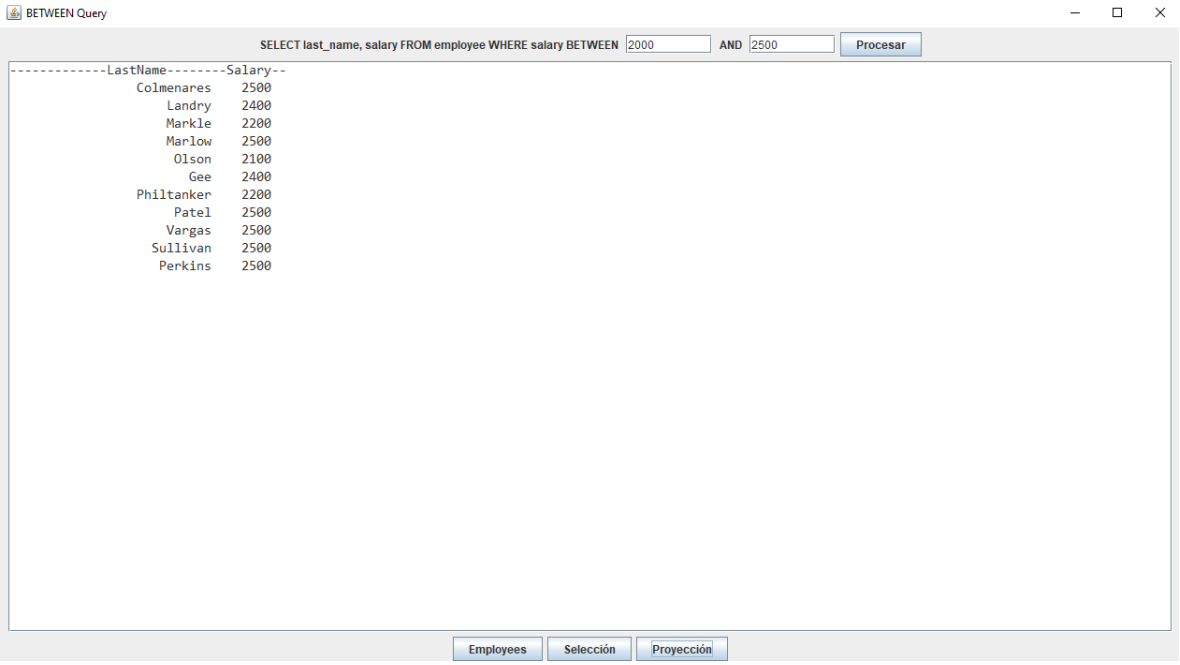
Botón Employees: Despliega la tabla de datos completa**.**



Botón Selección: Despliega las tuplas de los datos que se encuentran dentro del rango.



Botón Proyección: Muestra la proyección de los campos pertenecientes a las tuplas que cumplieron el rango dado por el usuario.



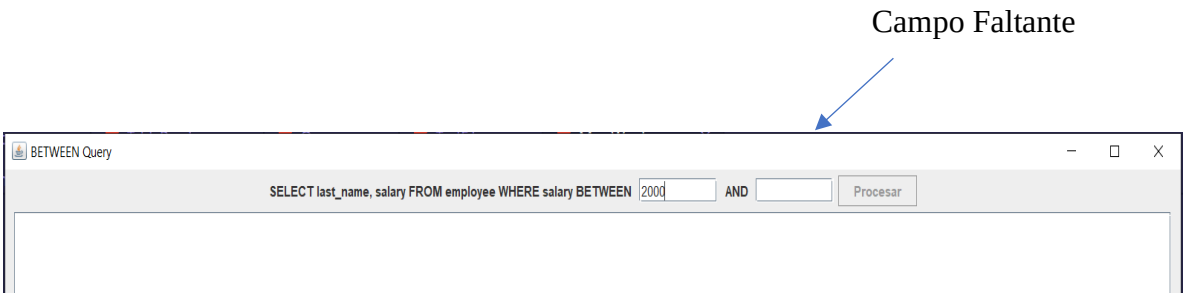
Excepciones

El software está diseñado con el objetivo de brindar calidad y utilidad al usuario; dicho programa informático detecta errores que el cliente pueda generar y le proporciona la solución necesaria.

1) Rango Incorrecto:
   Descripción: Error generado si el valor de fin de rango es menor al valor de inicio de rango.
   Solución: Colocar rango valido.
   Inicio de Rango < Fin de Rango
   Mensaje mostrado en pantalla: “El límite superior del salario debe ser mayor al límite inferior.”

Rango Invalido

2) Falta de Información:
   Descripción: El programa informático no dejará empezar el proceso hasta que se proporcione toda la información necesaria.
   Solución: Llenar ambos campos.



CÓDIGO FUENTE

Descripción de las clases Employee, Query, TableReader, TextFile y MainWindow.

Clase Employee:

Descripción: Es la clase que representa los datos de un empleado de la tabla; almacena todos los datos en forma de cadena (String), de tal forma que se guardan todos los caracteres posibles del atributo (el rango es establecido por el descriptor), sin embargo, les da formato a los elementos que lo requieran cuando sean utilizados.

```
import java.util.ArrayList;

public class Employee {
    // Atributos
    // Todos son cadenas (String) y contienen el máximo de longitud del atributo
    private String employeeID; // Identificador de Empleado
    private String firstName; // Apellido
    private String lastName; // Nombre de pila
    private String email; // Correo Electrónico
    private String phoneNumber; // Número de teléfono
    private String hireDate; // Fecha de contratación
    private String jobID; // Identificador de empleo
    private String salary; // Salario
    private String commissionPCT; // Porcentaje de comisiones (en decimales, puede no estar especificado)
    private String managerID; // Identificador de administrador (puede no estar especificado)
    private String departmentID; // Identificador de departamento de trabajo

    // Constructor
    // Recibe un conjunto de cadenas de texto ordenado, y de éste toma cada uno de los valores
    // para los atributos del empleado (en total, 11)
    public Employee(ArrayList<String> data) {
        employeeID = data.get(0);
        firstName = data.get(1);
        lastName = data.get(2);
        email = data.get(3);
        phoneNumber = data.get(4);
        hireDate = data.get(5);
        jobID = data.get(6);
        salary = data.get(7);
        commissionPCT = data.get(8);
        managerID = data.get(9);
        departmentID = data.get(10);
    }
```

```
    /* Métodos */
    // Retorna TODOS los atributos del empleado en una única cadena
    // Los atributos conservan su extensión con la que fueron introducidos (es decir, el tamaño
    // que es capaz de almacenar el atributo)
    @Override
    public String toString() {
        return employeeID + firstName + lastName + email + phoneNumber + hireDate + "        " + jobID + salary + " " + commissionPCT + managerID + " " + departmentID;
    }

    // Retorna el salario como número
    public int getSalaryAsInt(){
        return Integer.parseInt(salary.replace(" ", ""));
    }

    // Método de selección para la consulta
    // Retorna solo el apellido (lastName) y el salario (salary) en un una cadena.
    public String select() {
        String query = lastName.trim() + (lastName.trim().length() > 7 ? "\t\t" : "\t\t\t") + salary.trim();
        return query;
    }
}
```

Clase Query:

Descripción: Clase encargada de procesar la consulta con el rango ingresado por el usuario; almacena en tres tablas diferentes los resultados arrojados por el proceso paso a paso de la consulta.

Primer Paso: Obtención de la tabla completa.

Segundo Paso: Selección de las tuplas que cumplan el rango.

Tercer Paso: Proyección de los campos requeridos por el usuario de las tuplas seleccionadas.

Permite el retorno de estas tablas con el fin de mostrarlas en pantalla.

```
public class Query {
    // Tablas de la consulta, paso a paso
    private ArrayList<Employee> employees; // tabla completa
    private ArrayList<Employee> selectionTable; // tabla de la selección sobre la tabla completa
    private ArrayList<String> projectionTable; // tabla de la proyección sobre la selección

    // Procesa la consulta con condicional BETWEEN, en el rango específicado por los
    // límites en los parámetros
    public void between(int minsalary, int maxsalary) {
        TableReader tableReader = new TableReader(); // Lector de la tabla (utiliza el descriptor de archivos)

        // Inicializa las 3 tablas
        employees = new ArrayList<>();
        selectionTable = new ArrayList<>();
        projectionTable = new ArrayList<>();

        // Verifica que no haya error al abrir el archivo
        try {
            tableReader.readTable("Employees.txt");
            employees = tableReader.getEmployees(); // Paso 1: Selecciona la tabla completa
        }
        catch (Exception e) {
            // En caso de ocurrir algún error, termina la ejecución
            e.printStackTrace();
            System.exit(-1);
        }

        // Proceso de selección y proyección
        // Se busca que el salario esté dentro del intervalo (selección)
        for (Employee employee : employees) {
            if (employee.getSalaryAsInt() >= minsalary && employee.getSalaryAsInt() <= maxsalary) {
                selectionTable.add(employee); // Paso 2: Selección (Operación del Álgebra Relacional)
            }
        }

        // Se obtienen únicamente las columsnas solicitadas (proyección)
        for (Employee employee : selectionTable) {
            projectionTable.add(employee.select()); // Paso 3: Proyección (Operación del Álgebra Relacional)
        }
    }
```

```
    // Devuelve la tabla completa, en forma de cadena de texto, para ser desplegada en pantalla
    public String getEmployees() {
        StringBuilder employeeTable = new StringBuilder();
        for (Employee employee : employees) {
            employeeTable.append(employee.toString() + "\n");
        }
        return employeeTable.toString();
    }

    // Devuelve la tabla luego de la selección, en forma de cadena de texto, para ser desplegada en pantalla
    public String getSelectionTable() {
        StringBuilder selectedTable = new StringBuilder();
        for (Employee employee : selectionTable) {
            selectedTable.append(employee.toString() + "\n");
        }
        return selectedTable.toString();
    }

    // Devuelve la tabla luego de la proyección en la selección, en forma de cadena de texto, para ser desplegada en pantalla
    public String getProjectionTable() {
        StringBuilder projectedTable = new StringBuilder();
        for (String projection : projectionTable) {
            projectedTable.append(projection + "\n");
        }
        return projectedTable.toString();
    }
}
```

Clase TableReader:

Descripción: Es la clase encargada de leer correctamente el archivo que contiene la tabla, el fichero leído incluye el descriptor de archivo.

La clase descrita actualmente hace uso del descriptor, ya que lee los datos de la tabla y construye un objeto de clase Employee (clase que representa los datos de un empleado) para cada registro de la tabla.

Los registros son almacenados en una lista (ArrayList), la clase TableReader permite el retorno de la lista completa de objetos tipo Employee.

```
// Bibliotecas
import java.io.IOException;
import java.util.ArrayList;

public class TableReader {
    private String[] fileDescriptor; // Descriptor de Archivo
    private ArrayList<Employee> employees; // Registro de empleados

    // Constructor
    // Inicializa el registro de empleados
    public TableReader() {
        employees = new ArrayList<>();
    }

    // Lee el archivo de la tabla.
    // Convierte cada registro a un objeto de la clase Employee.
    // Guarda cada objeto en el registro de empleados (lista)
    public void readTable(String filename) throws IOException {
        // Se abre el archivo de la tabla
        TextFile file = new TextFile(); // Archivo fuente de la tabla
        file.openRead(filename);
        fileDescriptor = file.readLine().split(","); // se obtiene el descriptor de archivos

        String line = null;

        // Realiza la conversión de cada registro en la tabla en un objeto Employee
        // Utiliza el descriptor de archivo para realizar la división de la cadena leída del archivo
        while ((line = file.readLine()) != null) {
            ArrayList<String> data = new ArrayList<>();
            for (int i = 0; i < 11; i++) {
                // Límites para cortar la cadena de la tupla (obtenidos del descriptor)
                int lowBound = Integer.parseInt(fileDescriptor[i * 3 + 2]);
                int highBound = Integer.parseInt(fileDescriptor[i * 3 + 3]);

                data.add(line.substring(lowBound - 1, highBound)); // Añade cada segmento de cadena en una lista
            }
            employees.add( new Employee(data) ); // Convierte la lista de segmentos en un objeto Employee
        }
    }
```

```
    // Retorna la lista de empleados (en objetos Employee)
    public ArrayList<Employee> getEmployees() {
        return employees;
    }
}
```

Clase TextFile:

Descripción: Esta clase se encarga de la manipulación del archivo a leer y del fichero a escribir (abertura, recuperación de la información contenida en el fichero, escritura de información).

```
//Crean los lectores/escritores en buffer.
import java.io.BufferedReader;
import java.io.BufferedWriter;
//Crean los lectores/escritores de streams de entrada/salida
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
//Crean los streams de entrada/salida
import java.io.FileInputStream;
import java.io.FileOutputStream;
//Excepción de Entrada/Salida
import java.io.IOException;

public class TextFile{
  private BufferedReader reader; //Flujo de lectura
  private BufferedWriter writer;

  //Constructor sin apertura de archivo.
  public TextFile() {
    reader = null;
    writer = null;
  }

  //Abrir archivo para Lectura. Recibe el nombre del archivo e intenta abrirlo para lectura.
  //Retorna <true> si lo pudo abrir, <false> si no.
  public boolean openRead(String filename) {
    try{
      reader = new BufferedReader(new InputStreamReader(new FileInputStream(filename), "UTF-8"));
      return true;
    }
    catch(IOException ioe){
      return false;
    }
  }

  //Cierra archivo para Lectura. Cierra el archivo abierto en este objeto.
  //Retorna <true> si lo pudo cerrar, <false> si no.
  public boolean closeRead() {
    try{
      reader.close();
      return true;
    }
```

```
    catch(IOException ioe){
      return false;
    }
  }

  //Abrir archivo para Escritura. Recibe el nombre del archivo e intenta abrirlo para Escritura.
  //Retorna <true> si lo pudo abrir, <false> si no.
  public boolean openWrite(String filename) {
    try{
      writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(filename), "UTF-8"));
      return true;
    }
    catch(IOException ioe){
      return false;
    }
  }

  //Cierra archivo para Escritura. Cierra el archivo abierto en este objeto.
  //Retorna <true> si lo pudo cerrar, <false> si no.
  public boolean closeWrite() {
    try{
      writer.close();
      return true;
    }
    catch(IOException ioe){
      return false;
    }
  }

  //Leer Línea. Lee una línea del archivo de texto. Retorna la cadena o null,
  //según la función readLine(). Lanza la excepción IOException en caso de error.
  public String readLine() throws IOException {
    String s = reader.readLine();
    return s;
  }

  //Escribir Línea. Recibe un String y lo escribe al archivo, seguido de un salto de Línea
  //(line feed). Lanza una excepción IOException en caso de error.
  public void writeLine(String line) throws IOException {
    writer.write(line);
    writer.newLine();
    return;
```

Clase MainWindow:

Descripción: Clase donde se encuentran todos los componentes de la interfaz grafica en conjunto con su inicialización y actualización; MainWindow se encarga de mostrar las opciones de uso y los resultados obtenidos al usuario.

```
// Atributos de la ventana
private final int MIN_WIDTH = 960; // ancho mínimo de la ventana
private final int MIN_HEIGHT = 540; // alto mínimo de la ventana
private Query q; // objeto que realiza las consultas y obtiene los resultados paso a paso

// Componentes
private JPanel mainPanel; // panel principal

private JButton processButton; // botón de procesamiento de la consulta
private JButton viewEmployeeTableButton; // botón de visualización de la tabla completa
private JButton viewSelectionTableButton; // botón de visualización de la tabla luego de la seleción
private JButton viewProjectionTableButton; // botón de visualización de la tabla luego de la selección y la proyección

private JTextField minsalaryField; // campo de texto para recibir el límite inferior del salario
private JTextField maxsalaryField; // campo de texto para recibir el límite superior del salario

private JTextArea tableArea; // área de texto donde se mostrará la tabla seleccionada

// Constructor
// Inicializa la ventana y sus componentes
public MainWindow() {
    q = new Query();

    setTitle("BETWEEN Query");
    setExtendedState(getExtendedState() | JFrame.MAXIMIZED_BOTH);
    setMinimumSize(new Dimension(MIN_WIDTH, MIN_HEIGHT));
    setLocationRelativeTo(null);

    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent evt) {
            System.exit(0);
        }
    });

    init();

    setVisible(true);
}
```

```
/* Acciones en Eventos */
// Procesa la consulta con los valores especificados (al presionar el botón de "Procesar")
// El objeto "q" (de clase Query) genera las tablas correspondientes, solo si los límites son correctos
// La consulta no se procesa si el límite superior es menor al límite inferior
private void processQuery() {
    int minsalary = Integer.parseInt(minsalaryField.getText());
    int maxsalary = Integer.parseInt(maxsalaryField.getText());

    if (maxsalary < minsalary) {
        showMessage("\tEl límite superior del salario debe ser mayor al límite inferior.");
    }
    else {
        q.between(minsalary, maxsalary);
        setTable(2); // Inicialmente, muestra la tabla final (selección y proyección)
    }
}

// Cambia la tabla que se despliega en el área designada
private void setTable(int table) {
    String header; // cabecera que muestra los nombres de los atributros que se despliegan en pantalla
    switch (table) {
        case 0: // muestra la tabla completa
            header = "EmployeeID----FirstName-------------LastName-----------------Email---------------PhoneNumer---HireDate-------JobID-------Salary-Com-ManID-DeptID";
            tableArea.setText(header + "\n" + q.getEmployees());
            break;
        case 1: // muestra la tabla luego de la selección
            header = "EmployeeID----FirstName-------------LastName-----------------Email---------------PhoneNumer---HireDate-------JobID-------Salary-Com-ManID-DeptID";
            tableArea.setText(header + "\n" + q.getSelectionTable());
            break;
        case 2: // muestra la tabla luego de la selección y la proyección
            header = "LastName\t\tSalary";
            tableArea.setText(header + "\n" + q.getProjectionTable());
            break;
    }
    tableArea.setCaretPosition(0);

    viewEmployeeTableButton.setEnabled(true);
    viewSelectionTableButton.setEnabled(true);
    viewProjectionTableButton.setEnabled(true);
}
```