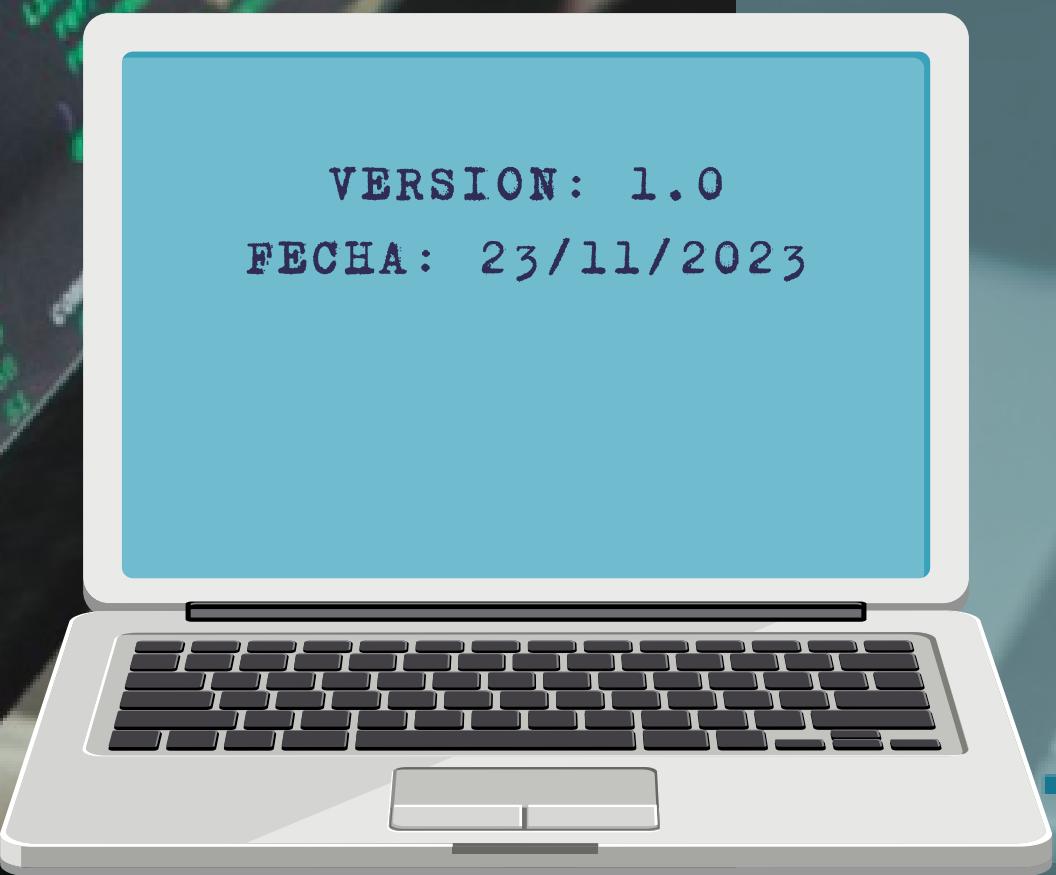


# MANUAL DEL PROGRAMADOR

AUTOR: JOSE LUIS OBIANG ELA NANGUANG

PROFE: JUAN ARIAS MASA

ASIGNATURA: SEGURIDAD DE LA INFORMACIÓN



## CIFRADO HILL..



# INDICE

1. JAVA DOC	3
2. DIAGRAMA DEL DISEÑO DEL PROGRAMA	4
3. FUENTES	5

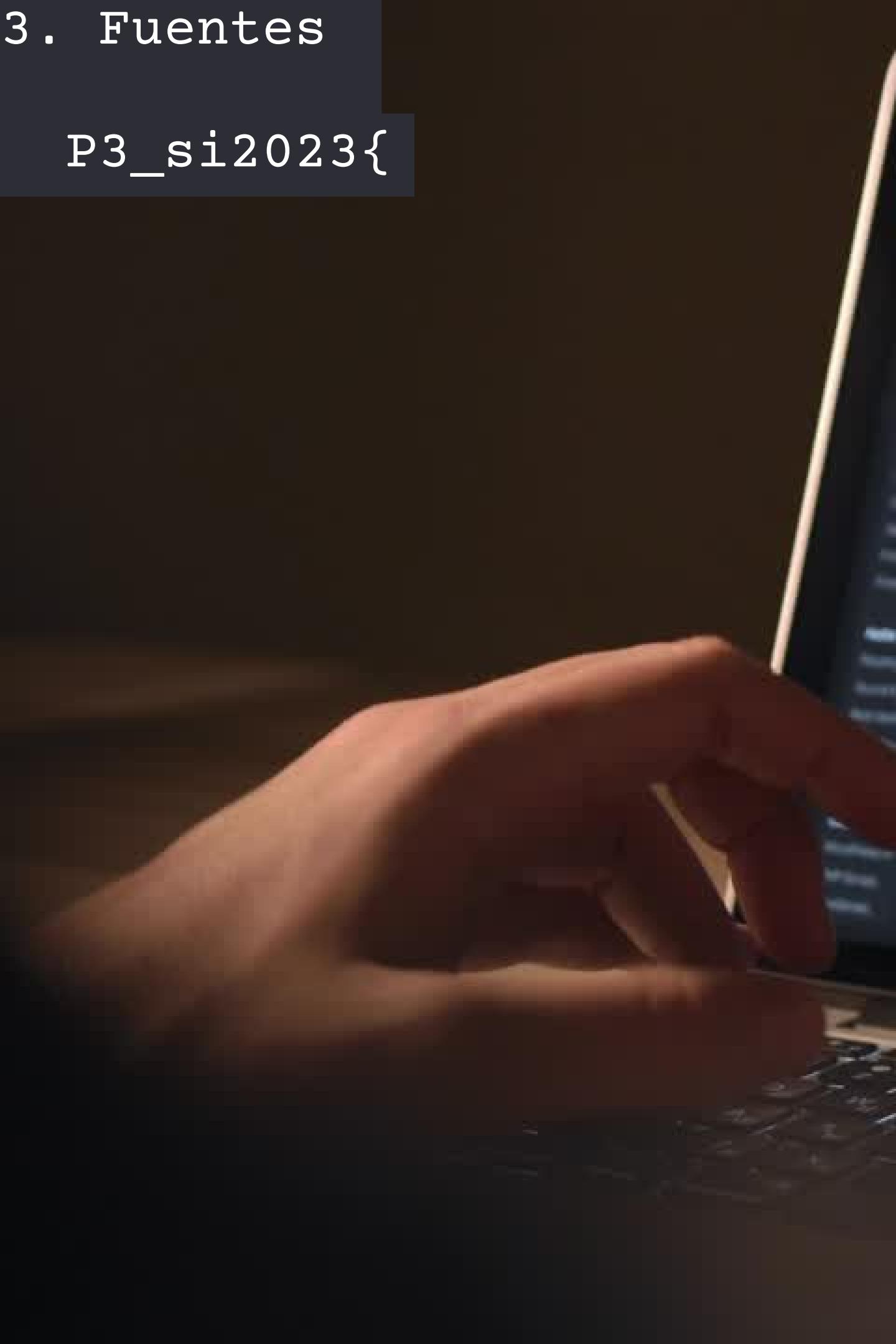
A dark-themed screenshot of a computer monitor. On the left, a file explorer sidebar shows several files and folders, including 'app/controllers', 'app/models', 'app/views', 'config', 'db', 'lib', 'public', and 'spec'. In the center, a terminal window displays a Ruby script, likely a test file, with syntax highlighting for code. The code includes require statements for 'spec\_helper', 'rspec/rails', 'capybara/rspec', and 'capybara/rails'. It also contains configuration for Capybara's javascript\_driver, Category.delete\_all, Shoulda::Matchers.configuration, config.integrate, and test framework setup. A note at the bottom suggests adding additional requirements. The right side of the screen is mostly black, indicating a dark environment.

```
4  # Prevent database truncation if the transaction fails
5  abort("The Rails environment is running in production mode!
6  require 'spec_helper'
7  require 'rspec/rails'
8
9  require 'capybara/rspec'
10 require 'capybara/rails'
11
12 Capybara.javascript_driver = :webkit
13 Category.delete_all; Category.create!(name: "Default")
14 Shoulda::Matchers.configure do |config|
15   config.integrate do |with|
16     with.test_framework :rspec
17     with.library :rails
18   end
19 end
20
21 # Add additional requires below this line if you need them
22
23 # Requires supporting ruby files with custom matchers and helpers
24 # in spec/support/ and its subdirectories. You can type "#load"
25 # in _spec.rb to require all matching feature-specific support
26 # files. For example, "#load feature_support/helpers.rb" would
27 # register :feature helper methods that can be used inside
28 # examples like "it" or "should". For more details see
29 # the RSpec documentation (www.relishapp.com/rspec)
30
31 # Encoding support for features
32 # Encoding.default_external = 'UTF-8'
```



## 2. DIAGRAMA

P3\_si2023{



```
import java.io.IOException;
import java.util.Scanner;

public class P3_si2023 {
    private static Scanner input = new Scanner(System.in);

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

        // Comprobar parametros de entrada
        Parametros parametros = new Parametros(args);
        Instancias instancias = new Instancias();

        EjecutarInstruccionesConfig runConfig = new EjecutarInstruccionesConfig(instancias, parametros);
        MostrarAyuda showH = new MostrarAyuda(instancias, parametros);

        showH.showH();

        if (parametros.isLessTwoParameters()) { // Si-> < 2 parametros

            if (parametros.isOneParameter()) { // Si->1 Parametro

                showH.showH();

            } else {

                parametros.isEmpty(runConfig.getInstancias().isEstadoTrazas()); // Sin parametros

            }

        } else { // >= 2 parametros

            if (parametros.isParameterF(runConfig.getInstancias().isEstadoTrazas())) { // Parametro 1: -f

                runConfig.runConfig();

            } else {
                Procesar.imprimirTrazasConSaltoLinea("Error, Parametro incorrecto",
                    runConfig.getInstancias().isEstadoTrazas());
            }

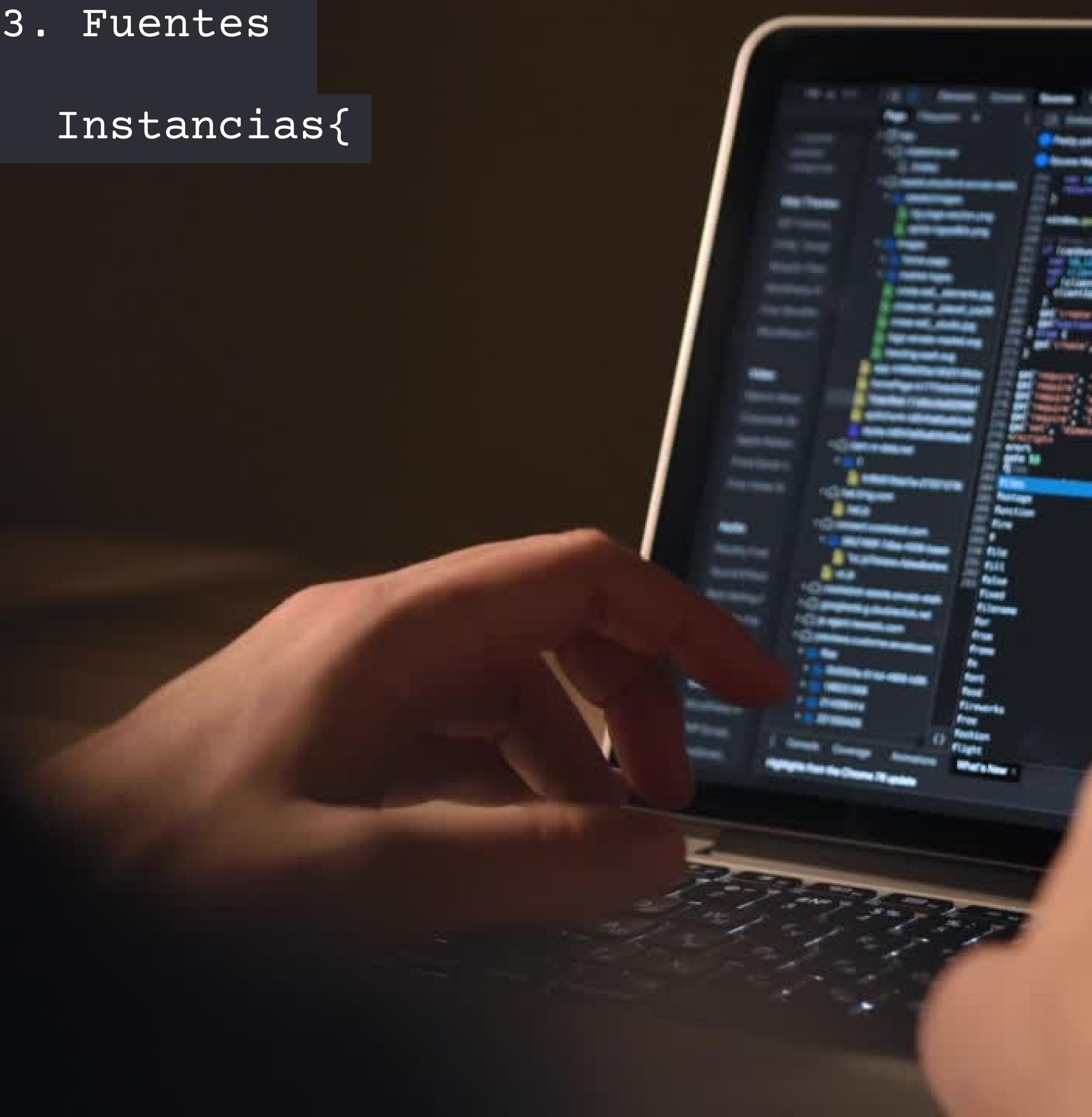
        }

        Procesar.imprimirTrazasConSaltoLinea("-----\n" +
            "Hasta una proxima ejecucion\n" +
            "-----\n", runConfig.getInstancias().isEstadoTrazas());
        System.out.println("FIN DEL PROGRAMA POR AUSENCIA DEL FICHERO DE CONFIGURACION");

    }

}
```

# Instancias{



```
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import java.io.File;

public class Instancias {
    private String pathInit;
    private String commandCodes;
    private boolean isRelleno;
    private SecretKey clave;
    private byte[] criptograma;
    private String textoEnClaro;
    private String textoEnClaroTemp;
    private String tipoCifrado;

    // Ficheros
    private File fileReadme;
    private File fileLogs;
    private File inputFile;
    private File outputFile;

    boolean estadoTraza; // Traza de activación: true->traza ON; false->traza OFF
    private String[] palabrasClave; // Palabras clave del fichero de configuración
    private IvParameterSpec ivParameterSpec;

    public Instancias() {
        this.pathInit = "src/main/resources/";
        this.isRelleno = true;
        this.clave = null;
        this.criptograma = null;
        this.extension = "";
        this.tipoCifrado = "AES";
        this.fileReadme = new File(pathInit + "README.txt"); // Fichero de ayuda
        this.fileLogs = new File(pathInit + "LOGS.txt"); // Fichero de logs
        this.fileKey = null; // Fichero clave
        this.inputFile = null; // Fichero entrada
        this.outputFile = null; // Fichero salida
        this.estadoTraza = true; // Traza de activación: true->traza ON; false->traza OFF
        this.palabrasClave = new String[]{"Genera_Clave", "Carga_Clave", "Fichero_Clave", "ficheroentrada",
            "ficherosalida", "AES", "CBC"}; // Palabras clave del fichero de configuración
        this.ivParameterSpec = null;
    }

    public String getPathInit() {
        return pathInit;
    }

    public void setPathInit(String pathInit) {
        this.pathInit = pathInit;
    }

    public String getCommandCodes() {
        return commandCodes;
    }

    public void setCommandCodes(String commandCodes) {
        this.commandCodes = commandCodes;
    }

    public boolean isRelleno() {
        return isRelleno;
    }

    public void setRelleno(boolean relleno) {
        this.isRelleno = relleno;
    }

    public SecretKey getKey() {
        return clave;
    }

    public void setKey(SecretKey clave) {
        this.clave = clave;
    }

    public byte[] getCriptograma() {
        return criptograma;
    }

    public void setCriptograma(byte[] criptograma) {
        this.criptograma = criptograma;
    }

    public String getTextoEnClaro() {
        return textoEnClaro;
    }

    public void setTextoEnClaro(String textoEnClaro) {
        this.textoEnClaro = textoEnClaro;
    }

    public String getExtension() {
        return extension;
    }

    public void setExtension(String extension) {
        this.extension = extension;
    }

    public String getTipoCifrado() {
        return tipoCifrado;
    }

    public void setTipoCifrado(String tipoCifrado) {
        this.tipoCifrado = tipoCifrado;
    }

    public File getFileReadme() {
        return fileReadme;
    }

    public void setFileReadme(File fileReadme) {
        this.fileReadme = fileReadme;
    }

    public File getFileLogs() {
        return fileLogs;
    }

    public void setFileLogs(File fileLogs) {
        this.fileLogs = fileLogs;
    }

    public File getFileKey() {
        return fileKey;
    }

    public void setFileKey(File fileKey) {
        this.fileKey = fileKey;
    }

    public File getInputFile() {
        return inputFile;
    }

    public void setInputFile(File inputFile) {
        this.inputFile = inputFile;
    }

    public File getOutputFile() {
        return outputFile;
    }

    public void setOutputFile(File outputFile) {
        this.outputFile = outputFile;
    }

    public boolean isEstadoTraza() {
        return estadoTraza;
    }

    public void setEstadoTraza(boolean estadoTraza) {
        this.estadoTraza = estadoTraza;
    }

    public String[] getPalabrasClave() {
        return palabrasClave;
    }

    public void setPalabrasClave(String[] palabrasClave) {
        this.palabrasClave = palabrasClave;
    }

    public IvParameterSpec getIvParameterSpec() {
        return ivParameterSpec;
    }

    public void setIvParameterSpec(IvParameterSpec ivParameterSpec) {
        this.ivParameterSpec = ivParameterSpec;
    }
```

### 3 . Fuentes

# EjecutarInstruccionesConfig

A close-up, low-angle shot of a person's hands typing on a laptop keyboard. The hands are illuminated by the screen's light, which also reflects off the keys. The background is dark.

# Parametros{

```

import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.util.Objects;

/**
 * Clase que maneja los parametros de ejecucion del programa
 * @author Jose Luis Obiang Ela Nanguang
 * @version 2.0
 */
public class Parametros {

    private final int totalParametros; //Numero de parametros introducidos
    private String[] parametros; //Array de parametros introducidos

    /**
     * Constructor por defecto: Inicializa todas las variables de la clase.
     */
    public Parametros() {
        this.totalParametros = parametros.length;
        this.parametros = new String[totalParametros];
    }

    /**
     * Constructor parametrizado
     * @param parametros, recibe un arreglo de String con los parametros
     */
    public Parametros(String[] parametros) {
        this.parametros = parametros;
        this.totalParametros = parametros.length;
    }

    /**
     * Devuelve el numero de parametros introducidos
     * @return total de parametros
     */
    public int getTotalParametros() {
        return totalParametros;
    }

    /**
     * Devuelve todos los parametros introducidos
     * @return todos los parametros introducidos
     */
    public String[] getParametros() {
        return parametros;
    }

    /**
     * Para mostrar informacion de la clase
     * @return
     */
    @Override
    public String toString() {
        return "Parametros{" +
            "totalParametros=" + totalParametros +
            ", parametros=" + Arrays.toString(parametros) +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Parametros that = (Parametros) o;
        return totalParametros == that.totalParametros && Arrays.equals(parametros, that.parametros);
    }

    @Override
    public int hashCode() {
        int result = Objects.hash(totalParametros);
        result = 31 * result + Arrays.hashCode(parametros);
        return result;
    }

    /**
     * Verifica si se ha introducido parametros
     */
}

```

```

* @return
*   <ul>
*     <li>true: si NO se ha introducido parametros </li>
*     <li>false: si se ha introducido parametros</li>
*   </ul>
*/
public boolean isEmpty(boolean estadoTraza) {
    if (totalParametros == 0) {
        Procesar.ImprimirTrazaConSaltoLinea("No has introducido ningun parametro", estadoTraza);
        return true;
    }
    return false;
}

/* Verifica si la cantidad de parametros es menor que 2
 */
public boolean isLessTwoParameters() {
    return totalParametros < 2;
}

/* Verifica si la cantidad de parametros es igual a 1
 */
public boolean isOneParameter() {
    return totalParametros == 1;
}

/* Verifica si el parametro es -h
 * @param estadoTraza Variable de tipo booleano, recibe el estado de la traza
 */
public boolean isParameterH(File fileReadme, boolean estadoTraza) throws IOException {
    if (parametros != null && parametros.length > 0) {
        if (parametros[0].equals("-h") || parametros[0].equals("--help")) {
            return true;
        }
    }
    return false;
}

/* Verifica si el parametro es -f
 * @param estadoTraza Variable de tipo booleano, recibe el estado de la traza
 */
public boolean isParameterF(boolean estadoTraza) {
    if (parametros != null && parametros.length > 0) {
        if (parametros[0].equals("-f") || parametros[0].equals("--fichero")) {
            return true;
        }
    }
    return false;
}

/* Verifica si el parametro es -l
 * @param estadoTraza Variable de tipo booleano, recibe el estado de la traza
 */
public boolean isParameterL(boolean estadoTraza) {
    if (parametros != null && parametros.length > 0) {
        if (parametros[0].equals("-l") || parametros[0].equals("--logs")) {
            return true;
        }
    }
    return false;
}

```

# Procesar{

```

import java.crypto.*;
import java.crypto.spec.IvParameterSpec;
import java.io.*;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Interfaz que se encarga de las operaciones de procesado de ficheros
 *
 * @author Jose Luis Oblang Ela Nanguag
 * @version 2.0
 */
public interface Procesar {

    /**
     * Lee un fichero y muestra su contenido
     *
     * @param fichero    recibe el fichero a leer
     * @param estadoTraza recibe el estado de la traza
     * @throws IOException
     */
    public static int leerFichero(File fichero, boolean estadoTraza) throws IOException {
        String linea = "";
        FileReader readme = null;
        BufferedReader lectorReadme = null;
        int totalLineas = 0;
        try {
            readme = new FileReader(fichero);
            lectorReadme = new BufferedReader(readme);
            while ((linea = lectorReadme.readLine()) != null) {
                if (fichero.getName().equalsIgnoreCase("logs.txt")) {
                    if (!linea.startsWith("//")) {
                        Procesar.imprimirTrazaConSaltoLinea("//" + linea.substring(0, linea.indexOf("//")) + ", " + estadoTraza);
                        totalLineas++;
                    } else if (linea.toLowerCase().contains("criptograma")) {
                        Procesar.imprimirTrazaConSaltoLinea("Texto en claro Formateado: " + linea.substring(linea.indexOf("//")));
                    }
                }
            }
            catch (FileNotFoundException e) {
                Procesar.imprimirTrazaSinSaltoLinea("No existe el fichero: " + fichero.getPath(), estadoTraza);
            }
            finally {
                if (lectorReadme != null) lectorReadme.close();
                if (readme != null) readme.close();
            }
        }
        return totalLineas;
    }

    /**
     * Comprueba si existe un fichero
     *
     * @param fichero Objeto de tipo File, recibe el fichero a comprobar
     * @return booleano
     * <ul>
     * <li>true: si existe el fichero</li>
     * <li>false: si no existe el fichero</li>
     * </ul>
     */
    public static boolean comprobarFichero(File fichero, String compare, boolean estadoTraza) {
        boolean fileExist = false;
        if (fichero != null) {
            if (fichero.exists()) // Comprobar si existe el fichero
                fileExist = true;
            else if (fichero.exists() && fichero.length() < 0) { // Si existe y no contiene nada
                Procesar.imprimirTrazaConSaltoLinea("El fichero " + fichero.getPath() + " existe pero esta vacio", estadoTraza);
                fileExist = true;
            }
        }
        return fileExist;
    }

    /**
     * Verifica si una linea esta vacia.
     *
     * @param linea Variable de tipo cadena, recibe la linea del contenido de un fichero
     * @return booleano
     * <ul>
     * <li>true si la linea esta vacia</li>
     * <li>false si la linea no esta vacia</li>
     * </ul>
     */
    public static boolean isEmptyLine(String linea) {
        return linea.trim().isEmpty();
    }

    /**
     * Verifica si el primer caracter de la linea es un comando.
     *
     * @param linea Variable de tipo cadena, recibe la linea del contenido de un fichero
     * @return null
     * <ul>
     * <li>true si el primer caracter de la linea es un comando, bandera o comentario</li>
     * <li>false si el primer caracter de la linea no es un comando, bandera o comentario</li>
     * </ul>
     */
    public static char lsCommandFlagComment(String linea, boolean estadoTraza) {
        char cfc = '';
        switch (linea.charAt(0)) {
            case '!' :
                cfc = '/';
                break;
            case '#' :
                cfc = '#';
                break;
            case ' ':
                cfc = ' ';
                break;
            default:
                break;
        }
        return cfc;
    }

    /**
     * Devuelve un fichero de entrada o salida.
     *
     * @param linea    Variable de tipo cadena, recibe la linea del contenido de un fichero
     * @param compare   Variable de tipo cadena, recibe el texto a comparar para saber si se trata de un fichero de entrada o de salida
     * @param estadoTraza Variable de tipo booleano, recibe el estado de la traza
     * @return fichero
     */
    public static File getFile(String linea, String compare, String pathInit, boolean estadoTraza) {
        int ultimoIndex = 0;
        String path = null;
        File fichero = null;
        if (compare.equalsIgnoreCase("ficherointrada")) {
            path = pathInit + getWordForPos(linea, 2);
        } else if (compare.equalsIgnoreCase("ficherosalida")) {
            path = pathInit + getWordForPos(linea, 2);
        } else if (compare.equalsIgnoreCase("clave")) {
            path = pathInit + getWordForPos(linea, 2);
        }
        fichero = new File(path); //Obtener fichero de E/S, formateado o fichero clave
    }

    /**
     * Comprueba si una linea contiene un texto especifico.
     *
     * @param linea    Variable de tipo cadena, recibe la linea del contenido de un fichero
     * @param compare   Variable de tipo cadena, recibe el texto a comparar
     * @param cf       Variable de tipo caracter, recibe el caracter comando o bandera
     * @return Devuelve
     * <ul>
     * <li>true: contiene la cadena pasada como parametro</li>
     * <li>false: NO contiene la cadena pasada como parametro</li>
     * </ul>
     */
    public static boolean containComparator(String linea, String compare, char cf) {
        String regex = cf + "\\$";
        String regex2 = cf + "\\$\\s+";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(linea);
        return matcher.find();
    }

    /**
     * Obtiene una palabra por posicion en una cadena
     *
     * @param sentence Variable de tipo cadena, recibe una cadena
     * @param pos      Variable de tipo entero, recibe la posicion de la cadena a obtener
     * @return
     */
    public static String getWordForPos(String sentence, int pos) {
        String[] words = sentence.trim().split("\\s+");
        try {
            return words[pos];
        } catch (ArrayIndexOutOfBoundsException e){
            System.out.println("No ha existido la clave de Usuario");
        }
        return "";
    }

    /**
     * Imprime mensajes en la consola con salto de linea segun este el estado de la traza.
     *
     * <ul>
     * <li>true: Imprime mensajes por consola</li>
     * <li>false: NO Imprime mensajes por consola</li>
     * </ul>
     */
    public static void imprimirFichaConSaltoLinea(String salida, boolean estadoTraza) {
        if (estadoTraza) {
            System.out.println(salida);
        }
    }

    /**
     * Copiar los bytes cifrados al array data manualmente
     *
     * @param j      Variable de tipo int, recibe el mensaje a mostrar por consola
     * @param estadoTraza Variable de tipo boolean, recibe el estado de la traza
     */
    public static void imprimirFichaConSaltoLinea(String salida, boolean estadoTraza) {
        if (estadoTraza) {
            System.out.println(salida);
        }
    }

    /**
     * Descifrar byte[] data, SecretKey clv, int[] vecInit throws NoSuchAlgorithmException,
     * NoSuchPaddingException, InvalidKeyException, BadPaddingException {
     *
     * @param data
     * @param clv
     * @param vecInit
     */
    public static String descifrar(byte[] data, SecretKey clv, int[] vecInit) throws NoSuchAlgorithmException,
        NoSuchPaddingException, InvalidKeyException, BadPaddingException {
        Cipher cifrador = Cipher.getInstance("AES/ECB/NoPadding");
        cifrador.init(Cipher.DECRYPT_MODE, clv);
        byte[] v = ajustarNumerosABbytes(vecInit);
        StringBuilder textoDescifrado = new StringBuilder();
        for (int l = 0; l < data.length; l += 16) {
            byte[] bloqueCifrado = new byte[16];
            System.arraycopy(data, l, bloqueCifrado, 0, 16);
            System.out.print("Bloque Cifrado " + (l / 16 + 1) + ": ");
            printBytes(bloqueCifrado);
            byte[] xor = null;
            try {
                xor = cifrador.doFinal(bloqueCifrado);
            } catch (IllegalBlockSizeException e) {
                System.out.println(">>> BLOQUE ILEGAL: El size introducido no es correcto.");
            }
            System.out.print("----- Criptograma: ");
            printBytes(xor);
        }
        return data;
    }

    /**
     * Copiar los bytes cifrados al array data manualmente
     *
     * @param j      Variable de tipo int, recibe el mensaje a mostrar por consola
     * @param estadoTraza Variable de tipo boolean, recibe el estado de la traza
     */
    public static void imprimirFichaConSaltoLinea(String salida, boolean estadoTraza) {
        if (estadoTraza) {
            System.out.println(salida);
        }
    }

    /**
     * Ajustar numeros a bytes
     *
     * @param numeros
     */
    public static byte[] ajustarNumerosABbytes(int[] numeros) {
        byte[] tv = new byte[16]; // Vector de inicializacion de 16 bytes
        for (int l = 0; l < numeros.length; l++) {
            tv[l] = new byte[16];
            for (int i = 0; i < a.length; i++) {
                result[i] = (byte) (a[i] ^ b[i]);
            }
        }
        return result;
    }

    /**
     * cifrarAES_CBC(String textoPlano, SecretKey secretKey, IvParameterSpec iv)
     * throws Exception {
     *
     * @param textoPlano
     * @param secretKey
     * @param iv
     */
    public static byte[] cifrarAES_CBC(String textoPlano, SecretKey secretKey, IvParameterSpec iv) {
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, iv);
        return cipher.doFinal(textoPlano.getBytes());
    }

    /**
     * descifrarAES_CBC(byte[] textoCifrado, SecretKey secretKey, IvParameterSpec iv)
     * throws Exception {
     *
     * @param textoCifrado
     * @param secretKey
     * @param iv
     */
    public static byte[] descifrarAES_CBC(byte[] textoCifrado, SecretKey secretKey, IvParameterSpec iv) {
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKey, iv);
        byte[] textoDescifrado = cipher.doFinal(textoCifrado);
        return new String(textoDescifrado);
    }
}

```

## MostrarAyuda{

```
import java.io.IOException;

public class MostrarAyuda {

    private Instancias instancias;
    private Parametros parametros;

    public MostrarAyuda(Instancias instancias, Parametros parametros){
        this.instancias = instancias;
        this.parametros = parametros;
    }

    public MostrarAyuda(){
        this.instancias = new Instancias();
        this.parametros = new Parametros();
    }

    public void showH() throws IOException {
        if (parametros.isParameterH(instancias.getFileReadme(), instancias.isEstadoTraza())) { // Si Unico
            parametro: -h
                Procesar.leerFichero(instancias.getFileReadme(), instancias.isEstadoTraza()); // Lectura fichero de
            ayuda
        } else if (parametros.isParameterF(instancias.isEstadoTraza())) { // Si Unico parametro: -f
                Procesar.imprimirTrazaConSaltoLinea("Falta el archivo de configuracion", instancias.isEstadoTraza());
        } else if (parametros.isParameterL(instancias.isEstadoTraza())) { // Si Unico parametro: -l
                Procesar.imprimirTrazaConSaltoLinea("LOGS:", instancias.isEstadoTraza());
                Procesar.printLogs(instancias.getFileLogs(), instancias.isEstadoTraza()); // Imprimir fichero Logs
        } else { // Parametro invalido: Permmittidos->-h, -f, -l
                Procesar.imprimirTrazaConSaltoLinea("Error, parametro incorrecto", instancias.isEstadoTraza());
                Procesar.imprimirTrazaConSaltoLinea("La sintaxis correcta es: P3_SI.2023 [-f config.txt] | [-h]", instancias.isEstadoTraza());
        }
    }
}
```



**MUCHAS  
GRACIAS**