

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

4-10-2017

PSP & PMDM

Práctica inicial java para PSP y
PMDM

Several thin, curved lines in dark blue and light grey originate from the bottom left and curve upwards and to the right.

Bryan Tibán Tigse

PROFESOR: FELIPE CABEZA LEIVA

Tabla de contenido

1	Introducción.....	2
2	Análisis y diseño.	2
2.1	ASI 1: Definición del Sistema.....	2
2.1.1	ASI 1.1: Determinación del alcance del sistema.	2
2.2	ASI 2: Establecimiento de requisitos.	2
2.2.1	ASI 2.1 Obtención de requisitos.....	2
2.2.2	Diagrama Entidad / Relación.	3
2.3	ASI 5: Análisis de clases.....	4
2.4	ASI 8: Definición de interfaces de usuario.....	4
2.4.1	JFVentanaPrincipal.....	4
2.4.2	Ventanas de confirmación de cambios y confirmación de toma de decisiones “críticas”.....	5
2.4.3	Ventana JDEditar.	5
3	Construcción	5
3.1	Creación de la BD.....	5
3.2	Clase Conexión.....	5
3.2.1	Constructor Conexión().	6
3.2.2	Método cerrarConexion().	6
3.2.3	Método cerrarConexion().	6
3.2.4	Método conectar().	6
3.2.5	Método Count().	7
3.2.6	Método buscarContacto();.....	8
3.2.7	Método aniadirContacto();.....	8
3.2.8	Método lisAlllistAllContactos().	9
3.2.9	Método vaciarAgenda().	9
3.2.10	Método insertarContactoEnFichero();.....	10
3.2.11	Método borrarContacto.....	10
3.2.12	Método commit().	11
3.2.13	Método Rollback.	11
3.2.14	Método modificarContacto().	11
4	Plan de pruebas.....	20
4.1	Pruebas de caja negra.	20

1 Introducción.

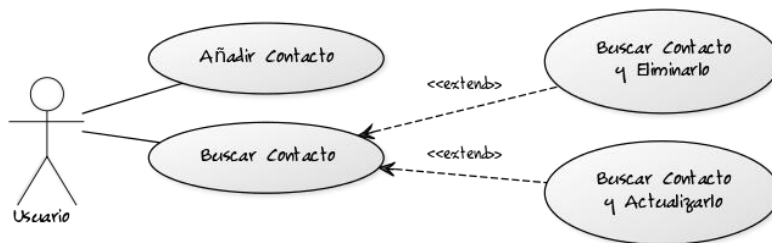
En este documento se detalla de manera específica lo que hará un software el cual su función es permitir al usuario interactuar con información de una agenda telefónica, se detalla cómo se ha llegado a conclusiones, como se llevan a cabo ideas de diseño del programa y también se reflejan las pruebas realizadas y el código de las mismas. Para la distribución de apartados y definiciones se ha intentado seguir el modelo de análisis y diseño de MétricaV3.

2 Análisis y diseño.

2.1 ASI 1: Definición del Sistema.

2.1.1 ASI 1.1: Determinación del alcance del sistema.

Se desea un programa el cual permita a un usuario gestionar una agenda telefónica, se desea que el usuario pueda interactuar a través de ventanas pudiendo realizar cuatro operaciones básicas: añadir, eliminar, identificar y modificar contacto, la GUI no debe estar sobrecargada por lo que debe ser lo más sencilla posible en apariencia, el programa gestionará la información a través de una conexión entre el código java y una base de datos alojada en Mysql workbench .Se aporta el siguiente caso de uso para mejor el entendimiento de sus funciones:



2.2 ASI 2: Establecimiento de requisitos.

2.2.1 ASI 2.1 Obtención de requisitos.

Requisitos Funcionales.	Requisitos NO Funcionales.
El usuario puede añadir contactos.	Se trabajará con una base de datos alojada en Mysql WorkBench.
Los contactos se guardarán en una Base de datos alojada en Mysql.	No se contempla la creación de varias agendas.
El usuario puede eliminar contactos.	El lenguaje de programación será java.
El usuario puede modificar un contacto ya existente.	El usuario realizará operaciones a través de 2 interfaces.
El usuario puede ver la lista de contactos.	Espacio mínimo en disco 1GB.

En el campo en el que se introducirá el nombre del contacto a crear, borrar o editar se permite la introducción de todo tipo de caracteres.	En el campo donde se introducirá el número telefónico del contacto a crear, borrar o editar no se permite la introducción de caracteres distintos a 0 a 9, es decir solo se admiten números.
Se deben pedir confirmaciones para la toma de decisiones.	La creación de la GUI queda a la libre decisión del programador, siempre y cuando permita hacer todas las tareas especificadas en el documento.
Se comprueba y crea una BD desde cero, todo transparente al usuario.	No se contempla que puedan haber contactos con el mismo nombre y número telefónico.
Se trasladan los datos generados por el programa creado a un fichero Excel, extensión .csv	No se admiten contactos sin número telefónico
	No se admiten contactos sin nombre

2.2.2 Diagrama Entidad / Relación.

Existe una única tabla en la que se alojan todos los contactos, se ha decidido establecer ambos campos como primary key habiendo descartado el uso de un campo auto numérico, de ahí las restricciones descritas en el apartado de "requisitos no funcionales"



2.3 ASI 5: Análisis de clases.



2.4 ASI 8: Definición de interfaces de usuario.

Existirán dos ventanas principales para ayudar al usuario en la manipulación de datos, se debe tener en cuenta que este programa no va dirigido a usuarios expertos por lo que la GUI debe ser lo más sencilla posible, también se hace uso de las interfaces predefinidas de java, clase JOPTIONPANE.

2.4.1 JFVentanaPrincipal.


- Esta ventana servirá para añadir nuevos contactos, buscar contactos mediante una tabla y editar contactos existentes en dicha tabla, contendrá en su apariencia lo siguiente:
- Barra de menú el cual al desplegarse muestre las opciones: SALIR y VACIAR AGENDA.
 - Opción SALIR: cerrará el programa e internamente salvará los datos en la base de datos.
 - Opción VACIAR AGENDA: limpiará la agenda de cualquier contacto existente hasta el momento.
- Debe contener dos campos:
 - Campo en el que se pueda introducir el nombre del contacto a crear.
 - Campo en el que se pueda introducir el número telefónico del contacto a crear.
 - Ambos campos son obligatorios.
- Tres botones:
 - Crear contacto: crea el contacto.
 - Borrar contacto: borra el contacto, se debe escribir en los campos el nombre y el número de teléfono a borrar, es obligatorio.
 - Editar contacto, se debe escribir en los campos el nombre y el número de teléfono a editar, es obligatorio. Este botón deriva a una nueva ventana, **VentanaEditar punto 2.4.2.**
- Contendrá una tabla en la que se listarán todos los contactos existentes.

2.4.2 Ventanas de confirmación de cambios y confirmación de toma de decisiones “críticas”.

- Se trabajará con la clase JOptionPane y las interfaces visuales de aviso, información, errores, confirmación... que trae predefinidas.
- Se mostrarán:
 - Al crear contacto, un mensaje informativo de que la creación se ha realizado correctamente, o de lo contrario un mensaje de error informando de la solución a aplicar si procede.
 - Al eliminar un contacto, se informará de la NO existencia del contacto si procede, se informará al usuario si los campos no están rellenos correctamente si procede, se pedirá confirmación para borrar un contacto ya que los cambios no serán reversibles.
 - Al salir, se pedirá confirmación para salir.
 - Al vaciar la agenda, se pedirá confirmación y dado que será necesario un reinicio de la aplicación (que se realiza automáticamente) se informará al usuario.

2.4.3 Ventana JEditor.

Esta ventana se abrirá al momento de ser llamada con la *el botón EDITAR* de la ventana Principal, **punto 1**. Contendrá la GUI que permitirá al usuario modificar el contacto previamente seleccionado, contendrá los siguientes elementos:

- Dado que esta ventana no se cerrará pulsando en la “x” de la esquina superior derecha de la ventana  la GUI debe contar con un botón o menú desplegable que permita volver a la ventana principal.
- Contendrá dos campos, que se deberán rellenar obligatoriamente.
- Campo nombre, obligatorio.
- Campo teléfono, obligatorio.
- Contendrá dos botones:
 - Botón confirmar: editará el elemento seleccionado, previa confirmación.
 - Botón Cancelar: vaciará los campos nombre y teléfono.

3 Construcción

3.1 Creación de la BD.

Teniendo en cuenta que lo único que se va a almacenar son dos campos, nombre y teléfono, y los cuales no pueden estar repetidos se hace uso de la siguiente orden DDL.

```
create database if not exists agenda;  
create table contactos( nombre varchar(50) not null, telefono varchar(50) not null,  
                        Constraint primary key PK_ID (nombre,telefono);
```

3.2 Clase Conexión.

Esta clase permite gestionar la conexión con la base de datos. Contiene métodos capaces de crear una un nuevo esquema en la base datos y métodos capaces de insertar nuevos elementos en la misma, como así también métodos capaces de modificar y eliminar elementos de la base de datos.

3.2.1 Constructor Conexion().

Método que permite crear la conexión, los pasos que sigue son los siguientes: llama al método conectar() el cual devuelve un número entero que es recibido por este método, si dicho número es igual a 1 quiere decir que la conexión con la BD agenda se realiza correctamente, si el resultado es 2 significa que la BD agenda no existe, por lo tanto este método Conexion() crea una nueva BD agenda.

3.2.2 Método cerrarConexion().

```
public Conexion() throws ClassNotFoundException, SQLException
{
    int opcion = conectar();
    String createBD = "create database if not exists agenda ";
    String createTable = "create table contactos( " +
        " nombre varchar(50) not null, " +
        " telefono varchar(50) not null," +
        " Constraint primary key PK_ID (nombre,telefono)" +
        ");";
    if (opcion == 2){
        try{
            terminal = con.createStatement();
            terminal.executeUpdate(createBD);
            terminal.executeUpdate("use agenda");
            terminal.executeUpdate(createTable);
            con.commit();
        }catch (SQLException e){
            e.printStackTrace();
        }
    }
}
```

3.2.3 Método cerrarConexion().

```
/**
 * Este método cierra la conexión previamente habiendo añadido en un fichero
 * ubicado en ./agenda.csv
 * todos los contactos existentes hasta el momento en la BD
 * @throws SQLException
 * @throws IOException
 */
public void cerrarConexion() throws SQLException, IOException
{
    insertarContactoEnFichero();
    con.close();
}
```

3.2.4 Método conectar().

```
/**
 * Este método intenta realizar dos conexiones, dependiendo de la BD a la que
 * se conecte retornará diferentes valores:
 * 1. La primera conexión se intenta realizar con la BD agenda si el resultado
 * es satisfactorio se retorna un entero 1
```

* 2. La segunda conexión se da cuando la primera resulta fallida, es decir cuando la BD agenda no existe, es entonces cuando

* la conexión se realiza a un esquema el cual no se puede eliminar en Mysql, que es el esquema TEST, si la conexión con esta BD resulta

* satisfactoria retorna un 2, según el entero retornado será el constructor de esta clase quien aplique las medidas correspondientes.

* @return se retorna un entero el cual sus valores son solo 1 o 2, si es 1 significa que la conexión con BD agenda ha sido satisfactoria

* si es 2 significa que la conexión con BD agenda ha fallado, será entonces cuando el constructor de esta clase tome las medidas adecuadas

* para crear la conexión con una nueva BD agenda.

*/

```
public int conectar()
{
    int ok = 0;
    try{
        Class.forName(driver);
        con = (Connection) DriverManager.getConnection(url, user, pwd);
        if(con != null){
            ok=1;
            con.setAutoCommit(false);
        }
    }catch (ClassNotFoundException e){
        // driver no encontrado
    }catch (SQLException e){
        try{
            this.url="jdbc:mysql://localhost/test";
            Class.forName(driver);
            con = (Connection) DriverManager.getConnection(url, user,
pwd);

            if(con != null){
                ok=2;
                con.setAutoCommit(false);
            }
        }catch (ClassNotFoundException e1){
            e1.printStackTrace();
        }catch (SQLException e1){
            e1.printStackTrace();
        }
    }
    return ok;
}
```

3.2.5 Método Count().

/**

* Este método devuelve el número de elementos existentes en la BD agenda.

* @return devuelve un entero Integer, el cual es el número de filas existentes en la BD

* @throws SQLException

*/


```

public int count() throws SQLException
{
    int res = 0;
    String query="select * from contactos";

    terminal = con.createStatement();
    ResultSet count = terminal.executeQuery(query);
    count.last();
    res =count.getRow();

    return res;
}

```

3.2.6 Método buscarContacto();

```

/**
 * Este método devuelve las filas ( la intención es que solo devuelva una fila)
 * que será el elemento el cual se está buscando,
 * para la búsqueda hace uso de parámetros.
 * @param nombre, este parámetro debe coincidir exactamente con el nombre del
 * contacto a buscar.
 * @param telefono, este parámetro debe coincidir exactamente con el teléfono
 * del contacto a buscar
 * @return retorna un ResultSet, el cual deberá ser tratado de manera que
 * permita certificar la existencia, o no existencia del contacto
 * @throws SQLException
 */
public ResultSet buscarContacto(String nombre, String telefono) throws
SQLException{
    String query ="select * from contactos where nombre like '"+nombre+"' AND
telefono like '"+telefono+"'";
    cursor = null;
    terminal = con.createStatement();
    terminal.executeUpdate("use agenda");
    cursor = terminal.executeQuery(query);

    return cursor;
}

```

3.2.7 Método anadirContacto();

```

/**
 * Este método permite añadir contactos a la base de datos
 * @param contacto, corresponde con el nombre del contacto a buscar
 * @return retorna un Boolean, el cual si resulta ser TRUE, significa que el
 * contacto se ha añadido correctamente,
 * y si resulta ser FALSE, significa que el usuario no se ha podido añadir a la
 * BD, generalmente significará que el contacto ya existe
 * @throws Exception
 * @throws CadenaDemasiadoLargaException, esta excepción se genera cuando el
 * nombre o telefono del contacto supera los 50 caracteres
 * @throws MySQLIntegrityConstraintViolationException, esta excepción se da
 * cuando se intenta añadir un contacto igual a uno ya existente

```

```

        */
        public boolean aniadirContacto(Contacto contacto) throws Exception,
CadenaDemasiadoLargaException, MySQLIntegrityConstraintViolationException{
            if(contacto.getNombre().length()>50 ||
contacto.getTelefono().length()>50)
            {
                throw new CadenaDemasiadoLargaException("La cadena debe ser menor o
igual a 50 caracteres");
            }
            String query ="insert into contactos values ( '" +
contacto.getNombre().toUpperCase() + "' , '" + contacto.getTelefono() + "' )";
            Statement sql = null;
            boolean ok = false;
            sql = con.createStatement();
            int sentencia = sql.executeUpdate(query);
            if(sentencia !=0){
                ok=true;
            }
            insertarContactoEnFichero();
            return ok;
        }

```

3.2.8 Método lisAlllistAllContactos().

```

/**
 * Este método lista todos los contactos.
 * @return retorna un ResultSet que deberá ser recorrido
 * @throws SQLException
 */
public ResultSet listAlllistAllContactos() throws SQLException{
    cursor = null;
    terminal = con.createStatement();
    cursor = terminal.executeQuery("select * from contactos");

    return cursor;
}

```

3.2.9 Método vaciarAgenda().

```

/**
 * Este método se encarga de vaciar la BD agenda y posteriormente con los
nuevos resultados llama al método inserContactoEnFichero()
 * @throws ClassNotFoundException
 * @throws SQLException
 */
public void vaciarAgenda() throws ClassNotFoundException, SQLException
{
    Class.forName(driver);
    DriverManager.getConnection("jdbc:mysql://localhost/test", "dam2",
"dam2");

    terminal = con.createStatement();
    terminal.executeUpdate("drop database if exists agenda");
    con.commit();
}

```

```

        try {
            insertarContactoEnFichero();
        } catch (IOException e) {
        }
    }
}

```

3.2.10 Método insertarContactoEnFichero();

```

/**
 * Este método inserta los contactos existentes hasta el momento en un fichero
con extensión .csv, "./agenda.csv"
 * @throws IOException
 * @throws SQLException
 */
public void insertarContactoEnFichero() throws IOException, SQLException{
    File fichero = new File("./agenda.csv");
    ResultSet elementos = listAlllistAllContactos();//listallcontactos
devuelve toda la tabla
    FileWriter fw = new FileWriter(fichero);
    BufferedWriter bw = new BufferedWriter(fw);
    //elementos.first();
    while(elementos.next()){// se recorre la tabla y se escribe en el fichero
        bw.write(elementos.getString("nombre")+" "+
elementos.getString("telefono"));
        bw.newLine();
    }
    bw.close();
}

```

3.2.11 Método borrarContacto

```

/**
 * Este método borra a priori, un solo contacto de la BD, los elementos
necesarios para el borrado se le pasa por parámetro diferentes
 * valores.
 * @param nombre, debe ser igual al nombre del contacto existente en la BD que
se quiere borrar.
 * @param telefono, debe ser igual al teléfono del contacto existente en la BD
que se quiere borrar.
 * @return retorna un entero. el cual si el borrado ha sido satisfactorio
devuelve un 1, en caso contrario retorna un 0.
 * @throws SQLException
 * @throws CadenaDemasiadoLargaException, esta excepción se da cuando por
parametro se le han pasado cadenas demasiado largas, superiores
 * a 50 caracteres.
 */
public int borrarContacto(String nombre, String telefono) throws SQLException,
CadenaDemasiadoLargaException
{
    nombre=nombre.toUpperCase();
    if(nombre.length()>50)

```

```

        {
            throw new CadenaDemasiadoLargaException("La cadena debe ser menor o
igual a 50 caracteres");
        }
        int line=0;
        String query ="Delete From contactos where nombre like '"+nombre+"' AND
telefono like '"+telefono+"'";

        terminal = con.createStatement();
        line = terminal.executeUpdate(query);

        return line;
    }

```

3.2.12 Método commit().

```

/**
 * Confirma los cambios realizados desde el último commit en la BD.
 * @throws SQLException
 */
public void commit() throws SQLException
{
    con.commit();
}

```

3.2.13 Método Rollback.

```

/**
 * Revierte los cambios realizados desde el último commit realizado.
 * @throws SQLException
 */
public void rollback() throws SQLException
{
    con.rollback();
}

```

3.2.14 Método modificarContacto().

```

/**
 * Este método modifica un contacto.
 * @param telefonoActual, este parámetro corresponde con el telefono del
contaco a modificar.
 * @param nombreActual,este parámetro corresponde con el nombre del contaco a
modificar.
 * @param telefonoNuevo, este parámetro corresponde con el NUEVO telefono del
contaco a modificar.
 * @param nombreNuevo, este parámetro corresponde con el NUEVO nombre del
contaco a modificar.
 * @return retorna un entero Integer, el cual si es 0 significará que la
actualización no se ha realizado, si retorna 1 significa que la actualización
 * se ha realizado correctamente.
 * @throws SQLException
 */

```

```

    public int modificarContacto(String telefonoActual, String nombreActual, String
telefonoNuevo, String nombreNuevo) throws SQLException
    {
        nombreNuevo.toUpperCase();
        int line=0;
        String query ="update contactos set  nombre= '" +nombreNuevo+"',
telefono = '" +telefonoNuevo+" where nombre like '" +nombreActual+

        "' AND telefono like '"+telefonoActual+"' ";

        terminal = con.createStatement();
        line = terminal.executeUpdate(query);
    }

```

Autor	Bryan Tibán	Fecha	04/10/2017	Versión	1.0
Aplicación	Agenda				
Identificador	CUID_01_AñadirContacto				
Prioridad	No Aplica	Urgencia	Media		
DESCRIPCIÓN BREVE *					
El objetivo es añadir contactos en la agenda, cerciorándose de la NO repetición del contacto.					
ACTORES *					
Usuario.					
Mysql Workbench.					
PRECONDICIONES *					
Precondición 1: Se debe realizar la conexión con la base de datos alojada en Mysql.					
Precondición 2: El programa pide datos existentes en la BD.					
FLUJO PRINCIPAL o SECUENCIA NORMAL					
Paso	Acción				
1	El usuario introduce nombre y número telefónico para un contacto.				
2	El programa añade el contacto a la base de datos alojada en Mysql.				
3	El programa informa al usuario de que el contacto se ha añadido correctamente y se confirma la inserción en la BD.				
FLUJO ALTERNATIVO o Excepciones *					
Paso	Acción				
1.1	Los campos están vacíos, se muestra mensaje informativo.				
2.1	El contacto ya existe, se muestra mensaje informativo.				
POSTCONDICIONES *					
Confirmar la inserción en la BD.					

EXTENSIONES
No se aplican.
INCLUSIONES
No se aplican
Rendimiento
<i>Comentario: El rendimiento no es óptimo</i>

Autor	Bryan Tibán	Fecha	04/10/2017	Versión	1.0
Aplicación	Agenda				
Identificador	CUID_02_BuscarContacto				
Prioridad	No Aplica	Urgencia	Media		
DESCRIPCIÓN BREVE *					
El objetivo es mostrar una lista con todos los contactos para que el usuario pueda buscar visualmente el contacto deseado.					
ACTORES *					
Usuario.					
Mysql Workbench.					
PRECONDICIONES *					
Precondición 1: Se debe realizar la conexión con la base de datos alojada en Mysql.					
Precondición 2: el programa pide los datos existentes en la BD.					
FLUJO PRINCIPAL o SECUENCIA NORMAL					
Paso	Acción				
1	El programa lista todos los contactos existentes en la BD.				
2	El usuario busca a través de la lista el contacto que desee.				
FLUJO ALTERNATIVO o Excepciones *					
Paso	Acción				
1.1	No existen datos en la BD, se muestra la lista vacía.				
2.1	El usuario no encuentra el contacto, no existe.				
POSTCONDICIONES *					
No se aplican.					
EXTENSIONES					

1. CUID_03_BuscarContactoYEliminarlo.
2. CUID_04_BuscarCOntratoYEditarlo.
INCLUSIONES
No se aplican
Rendimiento
<i>Comentario: El rendimiento no es óptimo</i>

Autor	Bryan Tibán	Fecha	04/10/2017	Versión	1.0
Aplicación	Agenda				
Identificador	CUID_03_BuscarContactoYEliminarlo				
Prioridad	No Aplica	Urgencia	Media		
DESCRIPCIÓN BREVE *					
El objetivo se da después de que el usuario haya identificado visualmente el contacto, una vez encontrado se procede a su eliminación.					
ACTORES *					
Usuario. Mysql Workbench.					
PRECONDICIONES *					
Precondición 1: Se debe realizar la conexión con la base de datos alojada en Mysql. Precondición 2: el programa pide los datos existentes en la BD.					
FLUJO PRINCIPAL o SECUENCIA NORMAL					
Paso	Acción				
1	El usuario identifica visualmente a través de una tabla el contacto que desea borrar				
2	El usuario rellena los campos con los datos del contacto a eliminar.				
3	El programa verifica la existencia del contacto				
4	El programa interactúa con la BD alojada en Mysql para eliminarlo				
5	El programa pide confirmación para el borrado.				
6	El programa borra el contacto de la BD y confirma los cambios.				
FLUJO ALTERNATIVO o Excepciones *					
Paso	Acción				
1.1	El usuario no existe.				
2.1	El usuario deja los campos vacíos, se informa de ello gráficamente.				

4.1	El contacto no existe, se informa de ello de manera gráfica.
5.1	El usuario cancela la transacción.
POSTCONDICIONES *	
Confirmar la transacción en la BD.	
EXTENSIONES	
No se aplican.	
INCLUSIONES	
No se aplican	
Rendimiento	
<i>Comentario: El rendimiento no es óptimo</i>	

Autor	Bryan Tibán	Fecha	04/10/2017	Versión	1.0
Aplicación	Agenda				
Identificador	CUID_04_BuscarYEditarContacto				
Prioridad	No Aplica	Urgencia	Media		
DESCRIPCIÓN BREVE *					
El objetivo se da después de que el usuario haya identificado visualmente el contacto, una vez encontrado se procede a su eliminación.					
ACTORES *					
Usuario.					
Mysql Workbench.					
PRECONDICIONES *					
Precondición 1: Se debe realizar la conexión con la base de datos alojada en Mysql.					
Precondición 2: el programa pide los datos existentes en la BD.					
FLUJO PRINCIPAL o SECUENCIA NORMAL					
Paso	Acción				
1	El usuario identifica visualmente a través de una tabla el contacto que desea editar				
2	El usuario rellena los campos con los datos del contacto a editar.				
3	El programa verifica la existencia del contacto				
4	El usuario introduce los nuevos datos para el contacto a editar.				
5	El programa interactúa con la BD alojada en Mysql para editarlo				
6	El programa pide confirmación para editar.				
7	El programa modifica el contacto de la BD y confirma los cambios.				
FLUJO ALTERNATIVO o Excepciones *					
Paso	Acción				
1.1	El usuario no existe.				

2.1	El usuario deja los campos vacíos, se informa de ello gráficamente.
4.1	El contacto no existe, se informa de ello de manera gráfica.
6.1	El usuario cancela la transacción.
POSTCONDICIONES *	
Confirmar la inserción en la BD.	
EXTENSIONES	
No se aplican.	
INCLUSIONES	
No se aplican	
Rendimiento	
<i>Comentario: El rendimiento no es óptimo</i>	

4 Plan de pruebas.

4.1 Pruebas de caja negra.

Prueba	Resultado esperado	Resultado obtenido
Insertar, eliminar, editar contacto. Nombre: <vacío> Teléfono: <vacío>	Error informativo	Error informativo
Insertar, eliminar, editar contacto. Nombre: String>50 caracteres. Teléfono: String>50 caracteres.	Error informativo	Error informativo
Insertar contacto que no existe aún. Nombre: pepe Teléfono: 1234	Creado correctamente	Creado correctamente
Insertar contacto que ya existe. Nombre: pepe Teléfono: 1234	Error informativo	Error informativo
Borrar contacto. Nombre: pepe Teléfono: 1234	Eliminado correctamente	Eliminado correctamente
Borrar contacto que no existe. Nombre: pepe Teléfono: 1234	Error informativo	Error informativo

Editar contacto que existe. Nombre: pepe Teléfono:1234	correctamente	correctamente
Editar contacto que no existe. Nombre: juan. Teléfono: 0000.	Error informativo	Error informativo
Función vaciar agenda con contactos.	correctamente	correctamente
Función vaciar agenda sin contactos.	correctamente	correctamente
Función volver, de JDialog.	correctamente	correctamente
Función salir.	correctamente	correctamente