Manual Técnico

Funcion login()

Se crea una función para ser llamada en caso que el usuario de clic en el botón de iniciar sesión o presione INTRO en una casilla de texto.

Se inicializan 3 variables: rol, userLogin, existe; las cuales serán re-definidas recorriendo el arreglo buscando el usuario solicitado, una vez encontrando el usuario solicitado, la variable existe, pasara a ser verdadera, luego verificara si el usuario tiene un rol administrador o un rol de usuario, mandando re-definiendo la variable rol y al mismo tiempo, re-definiendo la variable userLogin con la posición del usuario en el arreglo.

Una vez re-definidas las variables se comprueba que las casillas de texto tengan datos en el y utilizando las variables anteriores comprobamos su existencia, si el usuario y contraseña de esa posición son iguales a las que el usuario ingreso en la caja de texto y verificando su rol para ir a la vista administrativa o de usuario.

```
private void login() {
    AdminUsers au = new AdminUsers();

    String user = input_user.getText().toLowerCase();
    String pass = new String(input_pass.getPassword());

    String rol = "usuario";
    int userLogin = -1;
    boolean existe = false;

    for (int i = 0; i < au.usuariosDB.length; i++) {
        if (au.usuariosDB[i] != null) {
            if (au.usuariosDB[i].getUser().equals(user)) {
                existe = true;
                if (au.usuariosDB[i].getRol().equals("admin")) {
                    rol = "admin";
                    userLogin = i;
                } else if (au.usuariosDB[i].getRol().equals("usuario")) {
                    rol = "usuario";
                    userLogin = i;
                }
            }
        }
    }
    if (user.isBlank() || pass.isBlank() || user.equals("Nombre de usuario") || pass.equals("1234567890")) {
        JOptionPane.showMessageDialog(null, "Ingrese sus datos", "Iniciar Sesion", JOptionPane.WARNING_MESSAGE);
    } else {
        if (existe) {
            if (au.usuariosDB[userLogin].getUser().equals(user) && au.usuariosDB[userLogin].getPassword().equals(pass)) {
                if ("admin".equals(rol)) {
                    DashAdmin da = new DashAdmin();
                    this.dispose();
                    da.setVisible(true);
                } else if ("usuario".equals(rol)) {
                    DashUser du = new DashUser();
                    this.dispose();
                    du.userLogin(userLogin);
                    du.setVisible(true);
                }
            } else {
                JOptionPane.showMessageDialog(null, "Los datos no coinciden", "Iniciar Sesion", JOptionPane.WARNING_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(null, "El usuario no existe", "Iniciar Sesion", JOptionPane.ERROR_MESSAGE);
        }
    }

}
```

Se creo una función userLogin(int pos) para obtener la posición del usuario, dando a entender que el usuario está en línea, una vez mandada la posición del usuario, recorremos el arreglo verificando que la posición del usuario coincida con la del arreglo y definiendo la variable userLogin con la posición de ese arreglo.

Esta función se utiliza en toda la vista de usuario común y administrador.

```
    private int userLogin;

    public void userLogin(int pos) {
        AdminUsers au = new AdminUsers();
        for (int i = 0; i < au.usuariosDB.length; i++) {
            if (au.usuariosDB[i] != null) {
                if (au.usuariosDB[i].getUser().equals(au.usuariosDB[pos].getUser())) {
                    userLogin = i;
                    initComponents2();
                }
            }
        }

    }
```

En la vista del administrador de usuarios se crea un arreglo con 2 funciones, una que redimensiona el arreglo de usuariosDB y otra que verifica si aun tiene capacidad, si la capacidad del arreglo pasa el arreglo, se redimensiona el arreglo, aumentando su capacidad en 1 y creando un arreglo temporal para guardar los datos luego definirlos en el arreglo.

```
    public static db_users[] usuariosDB = new db_users[3];

    public static db_users[] redimUsers(db_users[] arreglo) {
        db_users[] temp = new db_users[usuariosDB.length + 1];
        for (int i = 0; i < usuariosDB.length; i++) {
            temp[i] = arreglo[i];
        }
        return temp;
    }

    public static boolean capacidadUsers(db_users[] arreglo) {
        boolean capacidades = false;
        for (int i = 0; i < arreglo.length; i++) {
            if (arreglo[i] == null) {
                capacidades = true;
                break;
            }
        }
        return capacidades;
    }
```

Para la edición de un usuario se creó una función para recibir la posición del usuario a modificar

```
    private int userEditNum;

    public void thisedit(int usuario) {
        this.userEditNum = usuario;
        initComponents2();
    }
```

Seleccionando la fila y obteniendo su id, si el id es igual al del arreglo manda la posición a la función thisedit(int usuario) y este se encarga de mostrar los datos del usuario en esa posición.

```
    private void btn_editUserActionPerformed(java.awt.event.ActionEvent evt) {
        EditUser editUser = new EditUser();
        int fila = table_users.getSelectedRow();

        if (fila >= 0) {
            long getPos = (long) table_users.getValueAt(fila, 0);
            for (int i = 0; i < usuariosDB.length; i++) {
                if (usuariosDB[i] != null) {
                    if (usuariosDB[i].getId() == getPos) {
                        editUser.thisedit(i);
                        this.dispose();
                        editUser.setVisible(true);
                        break;
                    }
                }
            }
        } else {
            JOptionPane.showMessageDialog(null, "Seleccione un usuario");
        }
    }
```

Para borrar un usuario se crea una función borrar, donde mandamos el arreglo y la posicion que deseamos borrar, en esta función creamos un arreglo temporal, obtenemos la nueva longitud y con Arraycopy copiamos el nuevo arreglo, definiendo nuestro arreglo usuariosDB como null y luego definiéndolo como el arreglo temporal.

```
        int fila = table_users.getSelectedRow();

        if (fila >= 0) {
            long id = (long) modelo.getValueAt(fila, 0);
            for (int i = 0; i < usuariosDB.length; i++) {
                if (usuariosDB[i] != null) {
                    if (usuariosDB[i].getId() == id) {
                        borrar(usuariosDB, i);
                        break;
                    }
                }
            }
        } else {
            JOptionPane.showMessageDialog(null, "Seleccione un usuario");
        }
    }

    private void borrar(db_users[] arr, int index) {
        db_users[] temp = new db_users[usuariosDB.length - 1];
        int restantes = usuariosDB.length - (index + 1);
        System.arraycopy(arr, 0, temp, 0, index);
        System.arraycopy(arr, index + 1, temp, index, restantes);
        usuariosDB = null;
        usuariosDB = temp;
        refresh();
    }
```

La función editUser() obtiene la información del usuario escritos en las casillas de texto y con un setter se cambia el dato en la posición que se mando anteriormente.

```
    private void editUser() {
        AdminUsers adminUsers = new AdminUsers();

        String _id = input_id.getText();
        long id = Long.parseLong(_id);
        String nombre = input_name.getText();
        String apellido = input_lastname.getText();
        String usuario = input_user.getText().toLowerCase();
        String pass1 = String.valueOf(input_password.getPassword());
        String pass2 = String.valueOf(input_password2.getPassword());
        int option_rol = input_rol.getSelectedIndex();

        String rol = "usuario";
        if (option_rol == 0) {
            rol = "usuario";
        } else if (option_rol == 1) {
            rol = "admin";
        }

        if (pass1.equals(pass2)) {

            adminUsers.usuariosDB[userEditNum].setName(nombre);
            adminUsers.usuariosDB[userEditNum].setLastname(apellido);
            adminUsers.usuariosDB[userEditNum].setPassword(pass1);
            adminUsers.usuariosDB[userEditNum].setRol(rol);

            JOptionPane.showMessageDialog(null, "Usuario editado", "Guardado!", JOptionPane.INFORMATION_MESSAGE);

            this.dispose();
            adminUsers.refresh();
            adminUsers.setVisible(true);

        } else {
            JOptionPane.showMessageDialog(null, "Las contraseñas no son iguales", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
```

Para la creación de nuevo usuario se creó la función newuser() que se llama cuando el usuario de clic o presione INTRO, esta función verifica que todos los datos hayan sido escritos, no haya espacios, las contraseñas sean iguales y que el usuario y ID no existan en la base de datos, una vez verificado todo, se encarga de guardarlo en el arreglo usuariosDB y refrescar la tabla.

```
            input_password.setText("password123");
            input_password2.setText("password123");
        }
    } else if (_id.equals("010203040506") || nombre.equals("Nombre...") || apellido.equals("Apellido...") || usuario.equals("Usuario...") || pass1.equals("pass
        JOptionPane.showMessageDialog(null, "Datos vacios", "Error", JOptionPane.ERROR_MESSAGE);
        if (pass1.equals("password123") || pass2.equals("password123")) {
            input_password.setText("password123");
            input_password2.setText("password123");
        } else if (pass1.isEmpty() || pass2.isEmpty()) {
            input_password.setText("password123");
            input_password2.setText("password123");
        }
    } else {
        if (pass1.equals(pass2)) {
            for (int i = 0; i < adminUsers.usuariosDB.length; i++) {
                if (adminUsers.usuariosDB[i] == null) {
                    for (int j = 0; j < adminUsers.usuariosDB.length; j++) {
                        if (adminUsers.usuariosDB[j].getUser().equals(usuario)) {
                            JOptionPane.showMessageDialog(null, "Ya existe el usuario en la base de datos", "Error", JOptionPane.ERROR_MESSAGE);
                            break;
                        } else {
                            if (adminUsers.usuariosDB[j].getId() == id) {
                                JOptionPane.showMessageDialog(null, "Ya existe el ID en la base de datos", "Error", JOptionPane.ERROR_MESSAGE);
                                break;
                            } else {
                                adminUsers.usuariosDB[i] = new db_users(id, nombre, apellido, usuario, pass1, rol);
                                JOptionPane.showMessageDialog(null, "Usuario creado con exito", "Guardado!", JOptionPane.INFORMATION_MESSAGE);
                                adminUsers.refresh();
                                this.dispose();
                                adminUsers.setVisible(true);
                                break;
                            }
                        }
                    }
                    break;
                }
            }
        } else {
            if (pass1.equals("password123") || pass2.equals("password123")) {
                input_password.setText("password123");
                input_password2.setText("password123");
            } else if (pass1.isEmpty() || pass2.isEmpty()) {
                input_password.setText("password123");
                input_password2.setText("password123");
            }
            JOptionPane.showMessageDialog(null, "Las contraseñas no son iguales", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
```

En la vista del administrador de bibliografías se crearon arreglos para 5 constructores los cuales son:

db_books: Todos los libros

db_libros: Solo libros

db_revistas: Solo revistas

db_tesis: solo tesis

db_digitales: solo libros digitales

asi también creando su función para redimensionar

```
    // ARREGLOS DE LIBROS
    public static db_books[] booksDB = new db_books[2];
    public static db_libros[] librosDB = new db_libros[2];
    public static db_revistas[] revistasDB = new db_revistas[2];
    public static db_tesis[] tesisDB = new db_tesis[2];
    public static db_digitales[] digitalesDB = new db_digitales[2];

    //REDIMENSIONAR ARREGLOS
    // ---------- REDIMENSIONAR LISTA DE BOOKS ----------
    public static db_books[] redimBooks(db_books[] arreglo) {
        db_books[] temp = new db_books[booksDB.length + 1];
        for (int i = 0; i < booksDB.length; i++) {
            temp[i] = arreglo[i];
        }
        return temp;
    }

    public static boolean capacidadBooks(db_books[] arreglo) {
        boolean capacidades = false;
        for (int i = 0; i < arreglo.length; i++) {
            if (arreglo[i] == null) {
                capacidades = true;
                break;
            }
        }
        return capacidades;
    }

    // ----------- REDIMENSIONAR LISTA DE LIBROS ----------
    public static db_libros[] redimLibros(db_libros[] arreglo) {
        db_libros[] temp = new db_libros[librosDB.length + 1];
        for (int i = 0; i < librosDB.length; i++) {
            temp[i] = arreglo[i];
        }
        return temp;
    }

    public static boolean capacidadLibros(db_libros[] arreglo) {
        boolean capacidades = false;
        for (int i = 0; i < arreglo.length; i++) {
            if (arreglo[i] == null) {
```

La función refresh() se encargara de verificar la capacidad de los arreglos asi como redimensionarlos.

```
// ---------- FUNCIONES ----------
public void refresh() {
    initComponents2();
    if (!capacidadBooks(booksDB)) {
        booksDB = redimBooks(booksDB);
    }
    if (!capacidadLibros(librosDB)) {
        librosDB = redimLibros(librosDB);
    }
    if (!capacidadRevistas(revistasDB)) {
        revistasDB = redimRevistas(revistasDB);
    }
    if (!capacidadTesis(tesisDB)) {
        tesisDB = redimTesis(tesisDB);
    }
    if (!capacidadDigitales(digitalesDB)) {
        digitalesDB = redimDigitales(digitalesDB);
    }

    return;
}
```

En la vista de la carga masiva, se crea una función generarCodeBook() que se encarga de generar un código único al libro, para tener un identificador.

```
private static String generarCodeBook() {
    Random rnm = new Random();
    boolean original = false;
    int code = (int) (rnm.nextDouble() * 9999 + 100);
    String codeBook = "Book" + code;

    while (!original) {
        for (int i = 0; i < AdminBooks.booksDB.length; i++) {
            if (AdminBooks.booksDB[i] == null) {
                codeBook = "Book" + code;
                original = true;
                break;
            } else {
                code = (int) (rnm.nextDouble() * 9999 + 100);
            }
        }
    }
    return codeBook;
}
```

Al darle clic en cargar se comprueba que el textarea no este vacío, si este contiene datos crea arreglos por cada salto de línea encontrando, que esto lo identificamos como libros, creando un ciclo de 0 a la cantidad de libros encontrados, luego crea un nuevo arreglo por cada punto y coma encontrados, identificándolos como los datos de cada libro y nuevamente creando un ciclo de 0 a la cantidad dato de los libros, dentro de el ingresamos un try catch para los errores y seguidamente definimos el formato de los datos como:

Tipo ; Autor ; Año ; ISBN ; Titulo ; Edición ; pClave ; Descripción ; Temas ; Copias ; Categorías ; Ejemplares ; Área ; Disponibles ;

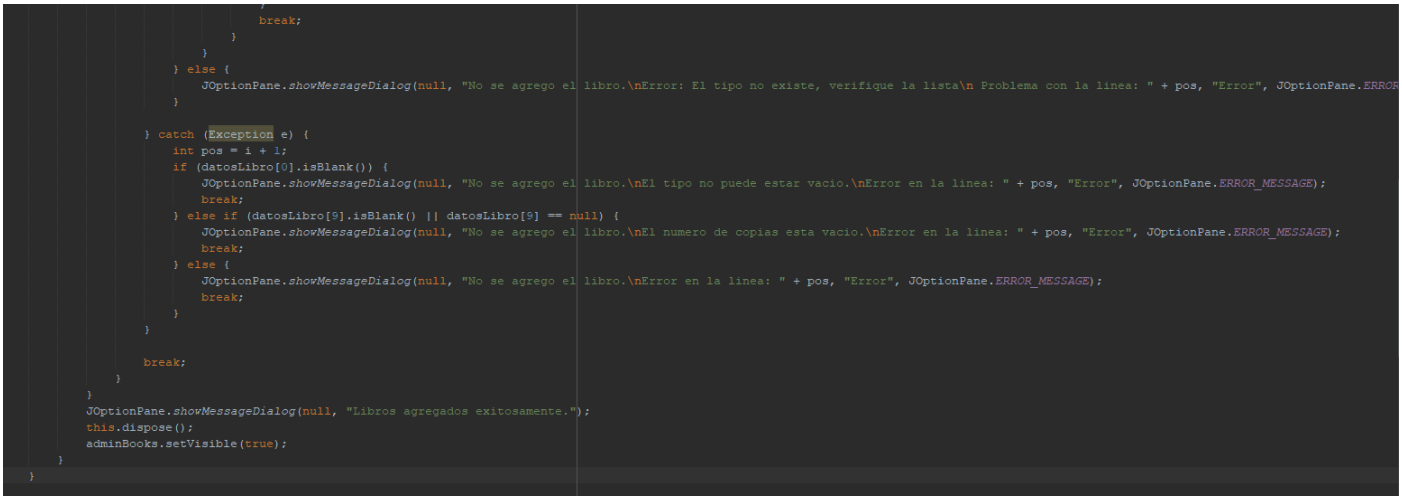
Y convirtiendo los datos a sus respectivos tipos, creando un if, para el ISBN, en caso que este este vacío, su valor será de 0 y si tiene algún dato, obtendrá el dato del arreglo.

```
String carga = input_areaCarga.getText();

if (carga.isBlank()) {
    JOptionPane.showMessageDialog(null, "El campo esta vacio", "Error", JOptionPane.ERROR_MESSAGE);
} else {
    String _cadenaTexto = input_areaCarga.getText();
    String[] Libros = _cadenaTexto.split("\n");
    for (int i = 0; i < Libros.length; i++) {
        String[] datosLibro = Libros[i].split(";");
        for (int j = 0; j < datosLibro.length; j++) {

            try {
                int tipo = Integer.parseInt(datosLibro[0]);
                String autor = datosLibro[1];
                String year = datosLibro[2];
                String _isbn = datosLibro[3];
                long isbn1= 0;
                if (_isbn.isBlank()) {
                    isbn1 = 0;
                }else{
                    isbn1 = Long.parseLong(datosLibro[3]);
                }

                String titulo = datosLibro[4];
                String edicion = datosLibro[5];
                String _pclave = datosLibro[6];

                String[] pclave = _pclave.split(",");
                String desc = datosLibro[7];
                String _temas = datosLibro[8];
                String[] temas = _temas.split(",");
                int copias = Integer.parseInt(datosLibro[9]);

                String categoria = datosLibro[10];
                String ejemplares = datosLibro[11];
                String area = datosLibro[12];
                int disponibles = Integer.parseInt(datosLibro[13]);

                int pos = i + 1;
```

Luego buscamos el tipo y recorremos los arreglos de cada tipo insertando los datos en cada arreglo

```
                int pos = i + 1;

                if (tipo == 0) {
                    for (int book = 0; book < adminBooks.booksDB.length; book++) {
                        if (adminBooks.booksDB[book] == null) {
                            // GENERAR CODIGO DEL LIBRO
                            String codeBook = generarCodeBook();
                            // AGREGAR BOOK
                            for (int libro = 0; libro < adminBooks.librosDB.length; libro++) {
                                if (AdminBooks.librosDB[libro] == null) {
                                    AdminBooks.booksDB[book] = new db_books(codeBook, titulo, autor, desc, year, pclave, edicion, temas, copias, disponibles, tipo);
                                    AdminBooks.librosDB[libro] = new db_libros(codeBook, autor, year, isbnl, titulo, edicion, pclave, desc, temas, copias, disponibles, tipo);
                                    adminBooks.refresh();
                                    break;
                                }
                            }
                            break;
                        }
                    }
                } else if (tipo == 1) {
                    for (int book = 0; book < adminBooks.booksDB.length; book++) {
                        if (adminBooks.booksDB[book] == null) {
                            // GENERAR CODIGO DEL LIBRO
                            String codeBook = generarCodeBook();
                            // AGREGAR BOOK
                            for (int revista = 0; revista < adminBooks.revistasDB.length; revista++) {
                                if (adminBooks.revistasDB[revista] == null) {
                                    AdminBooks.booksDB[book] = new db_books(codeBook, titulo, autor, desc, year, pclave, edicion, temas, copias, disponibles, tipo);
                                    adminBooks.revistasDB[revista] = new db_revistas(codeBook, autor, year, titulo, edicion, desc, categoria, ejemplares, temas, pclave, copias, disponibl
                                    adminBooks.refresh();
                                    break;
                                }
                            }
                            break;
                        }
                    }
                } else if (tipo == 2) {
                    for (int book = 0; book < adminBooks.booksDB.length; book++) {
                        if (AdminBooks.booksDB[book] == null) {
                            // GENERAR CODIGO DEL LIBRO
                            String codeBook = generarCodeBook();
                            // AGREGAR BOOK
                            for (int tesis = 0; tesis < adminBooks.tesisDB.length; tesis++) {
                                if (adminBooks.tesisDB[tesis] == null) {
                                    AdminBooks.booksDB[book] = new db_books(codeBook, titulo, autor, desc, year, pclave, edicion, temas, copias, disponibles, tipo);
```

Y en el catch se verifica si el tipo esta vacío, soltara un error, si las copias esta vacío soltara un error y si hay otro error soltara un mensaje de error.



```
                                break;
                            }
                        }
                    } else {
                        JOptionPane.showMessageDialog(null, "No se agrego el libro.\nError: El tipo no existe, verifique la lista\n Problema con la linea: " + pos, "Error", JOptionPane.ERROR
                    }

                } catch (Exception e) {
                    int pos = i + 1;
                    if (datosLibro[0].isBlank()) {
                        JOptionPane.showMessageDialog(null, "No se agrego el libro.\nEl tipo no puede estar vacio.\nError en la linea: " + pos, "Error", JOptionPane.ERROR_MESSAGE);
                        break;
                    } else if (datosLibro[9].isBlank() || datosLibro[9] == null) {
                        JOptionPane.showMessageDialog(null, "No se agrego el libro.\nEl numero de copias esta vacio.\nError en la linea: " + pos, "Error", JOptionPane.ERROR_MESSAGE);
                        break;
                    } else {
                        JOptionPane.showMessageDialog(null, "No se agrego el libro.\nError en la linea: " + pos, "Error", JOptionPane.ERROR_MESSAGE);
                        break;
                    }
                }

                break;
            }
        }
        JOptionPane.showMessageDialog(null, "Libros agregados exitosamente.");
        this.dispose();
        adminBooks.setVisible(true);
    }
}
```

La función editBook() tiene la misma función que editUser(), con el cambio que se manda el código del libro y se recorre el arreglo de bookDB y dependiendo de su tipo se dirige a las funciones para insertar datos en las cajas de texto

```
23      private int codeDigital;
24
25      public void editBook(String codeBook) {
26          AdminBooks adminBooks = new AdminBooks();
27
28          for (int i = 0; i < adminBooks.booksDB.length; i++) {
29              if (adminBooks.booksDB[i] != null) {
30                  if (adminBooks.booksDB[i].getCodeBook().equals(codeBook)) {
31                      editCodeBook = adminBooks.booksDB[i].getTipo();
32                      input_type.setSelectedIndex(editCodeBook);
33                      codeBooks = i;
                        if (editCodeBook == 0) {
35                          DataLibro();
36                      } else if (editCodeBook == 1) {
37                          DataRevista();
38                      } else if (editCodeBook == 2) {
39                          DataTesis();
40                      } else if (editCodeBook == 3) {
41                          DataDigital();
42                      }
43                      break;
44                  }
45              }
46
47          }
48          for (int i = 0; i < adminBooks.librosDB.length; i++) {
49              if (adminBooks.librosDB[i] != null) {
50                  if (adminBooks.librosDB[i].getCodeBook().equals(codeBook)) {
51                      codeLibro = i;
52                      break;
53                  }
54              }
55
56          }
57          for (int i = 0; i < adminBooks.revistasDB.length; i++) {
58              if (adminBooks.revistasDB[i] != null) {
```

Las funciones Data<tipo>() inserta los datos del libro en las cajas de texto

```
    public void DataLibro() {
        AdminBooks adminBooks = new AdminBooks();
        if (adminBooks.librosDB[codeLibro] != null) {
            input_autorL.setText(adminBooks.librosDB[codeLibro].getAutor());
            input_yearL.setText(String.valueOf(adminBooks.librosDB[codeLibro].getYear()));
            input_isbnL.setText(String.valueOf(adminBooks.librosDB[codeLibro].getIsbn()));
            input_tituloL.setText(adminBooks.librosDB[codeLibro].getTitulo());
            input_edicionL.setText(String.valueOf(adminBooks.librosDB[codeLibro].getEdicion()));
            input_copiasL.setText(String.valueOf(adminBooks.librosDB[codeLibro].getCopias()));
            input_disponiblesL.setText(String.valueOf(adminBooks.librosDB[codeLibro].getDisponibles()));

            String[] _pclave = adminBooks.librosDB[codeLibro].getP_clave();
            String pclave = String.join(",", _pclave);
            inputArea_Pclave.setText(pclave);
            inputArea_desc.setText(adminBooks.librosDB[codeLibro].getDesc());
            String[] _temas = adminBooks.librosDB[codeLibro].getTemas();
            String temas = String.join(",", _temas);
            inputArea_temas.setText(temas);
            input_codeBook.setText(adminBooks.booksDB[codeLibro].getCodeBook());
        }
    }

    public void DataRevista() {...23 lines }

    public void DataTesis() {...21 lines }

    public void DataDigital() {...26 lines }
```

La función Edit<tipo>() se encarga de verificar que las cajas de texto estén llegas y actualizar los arreglos

```
        String categoria = input_categoriaR.getText();
        String ejemplares = input_ejemplaresR.getText();
        String _copias = input_copiasR.getText();
        String _disponibles = input_disponiblesR.getText();

        if (desc.isBlank() || _pclave.isBlank() || _temas.isBlank() || autor.isBlank() || year.isBlank() || categoria.isBlank() || ejemplares.isBlank() || titulo.isBlank() || edicion.isBl
            JOptionPane.showMessageDialog(null, "No puede dejar campos vacios");
        } else {
            int copias = Integer.parseInt(_copias);
            int disponibles = Integer.parseInt(_disponibles);
            // ACTUALIZA LA BASE DE DATOS DE BOOKS
            adminBooks.booksDB[codeBooks].setTitulo(titulo);
            adminBooks.booksDB[codeBooks].setAutor(autor);
            adminBooks.booksDB[codeBooks].setDesc(desc);
            adminBooks.booksDB[codeBooks].setYear(year);
            adminBooks.booksDB[codeBooks].setP_clave(pclave);
            adminBooks.booksDB[codeBooks].setEdicion(edicion);
            adminBooks.booksDB[codeBooks].setCopias(copias);
            adminBooks.booksDB[codeBooks].setDisponibles(disponibles);
            adminBooks.booksDB[codeBooks].setTemas(tema);

            // ACTUALIZA LA BASE DE DATOS DE REVISTAS
            adminBooks.revistasDB[codeRevista].setAutor(autor);
            adminBooks.revistasDB[codeRevista].setYear(year);
            adminBooks.revistasDB[codeRevista].setTitulo(titulo);
            adminBooks.revistasDB[codeRevista].setEdicion(edicion);
            adminBooks.revistasDB[codeRevista].setCategoria(categoria);
            adminBooks.revistasDB[codeRevista].setEjemplares(ejemplares);
            adminBooks.revistasDB[codeRevista].setCopias(copias);
            adminBooks.revistasDB[codeRevista].setDisponibles(disponibles);
            adminBooks.revistasDB[codeRevista].setP_clave(pclave);
            adminBooks.revistasDB[codeRevista].setDesc(desc);
            adminBooks.revistasDB[codeRevista].setTemas(tema);

            adminBooks.refresh();
            this.dispose();
            adminBooks.setVisible(true);
        }

    }

    private void editTesis() {...52 lines }
```

Se crea una función actualizarDB que se encarga de actualizar todas las tablas pidiendo el código del libro y la disponibilidad.

```
    private void actualizarDB(String codeBook, int disponibles) {
        AdminBooks ab = new AdminBooks();

        for (int i = 0; i < ab.booksDB.length; i++) {
            if (ab.booksDB[i] != null) {
                if (ab.booksDB[i].getCodeBook().equals(codeBook)) {
                    ab.booksDB[i].setDisponibles(disponibles);
                }
            }
        }

        for (int i = 0; i < ab.librosDB.length; i++) {
            if (ab.librosDB[i] != null) {
                if (ab.librosDB[i].getCodeBook().equals(codeBook)) {
                    ab.librosDB[i].setDisponibles(disponibles);
                }
            }
        }
        for (int i = 0; i < ab.tesisDB.length; i++) {
            if (ab.tesisDB[i] != null) {
                if (ab.tesisDB[i].getCodeBook().equals(codeBook)) {
                    ab.tesisDB[i].setDisponibles(disponibles);
                }
            }
        }
        for (int i = 0; i < ab.revistasDB.length; i++) {
            if (ab.revistasDB[i] != null) {
                if (ab.revistasDB[i].getCodeBook().equals(codeBook)) {
                    ab.revistasDB[i].setDisponibles(disponibles);
                }
            }
        }

    }
```