Criterio C: Desarrollo

CLASES

Para manejar las medicinas dentro de la solución informática, se elaboró una clase Medicina, con las características especificadas por el usuario como variables de instancia. La clase Medicina cuenta con un constructor sin parámetros (utilizado en el código principal en caso de emergencias, cuando alguno de los parámetros de la medicina que se busca crear no es válido), un constructor con parámetros (usado para crear el resto de las medicinas) y sus métodos *getters* y *setters*, que permiten el encapsulamiento de las variables de instancia de la medicina.

```
public class MedicinaS {

  //Variables de instancia
  private String nombre;
  private String contenido; //Sales
  private Calendar caducidad;
  private boolean caduco;
  private String dosis;
  private String notas;

  //Constructor nulo
  public MedicinaS(){
    this.nombre = "XXXX";
  }

  //Constructor con parametros
  public MedicinaS(String nom, String cont, Calendar cad, boolean c, String cant, String not){
    this.nombre = nom;
    this.contenido = cont;
    this.caducidad = cad;
    caducidad.setLenient(false);
    this.caduco = c;
    this.dosis = cant;
    this.notas = not;
  }
```

Figura 1 - Fragmento de clase Medicina

Además de Medicina, se utilizan las clases de la librería javax.swing para elaborar la interface gráfica de la solución informática.

HERENCIA

Se utilizó la técnica de herencia en la solución informática para desarrollar los diferentes aspectos gráficos de la misma. Específicamente, los componentes (principalmente las ventanas del programa) heredan de los atributos y métodos de las clases originales, lo cual permite su manipulación a través de los métodos ya establecidos, pero me permite incorporar mis propias implementaciones a la clase. Se decidió crear una clase nueva que heredara las propiedades de una ya existente, y no simplemente crear una nueva instancia de esta clase ya establecida, para manejar el contenedor como una unidad, que se llamara desde diferentes puntos del programa. De esta manera, podía editar un componente y asegurar que sus atributos no se alterarán, debido al encapsulamiento de ésta. Ejemplos de estas herencias se pueden encontrar en FrameAgregar y FrameMain.

```
public class FrameAgregarS extends JFrame {
```

```
/////Frame/////////////////////////////////////////////
super();
setSize(590,350);
setLocationRelativeTo(null);
setBackground(new Color(23, 127, 179));
setLayout(new GridLayout(1,1));
setTitle("Agregar Medicinas");
setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
```

Figura 2 - Declaración FrameAgregar y fragmento

```
public class FrameMainS extends JPanel{
```

```
public FrameMainS(){

  //Inicializacion y propiedades de ventana
  super();
  setPreferredSize(new Dimension(630,370));
```

Figura 3 - Declaración FrameMain y fragmento

POLIFORMISMO

Se utilizó la técnica de poliformismo para modificar una implementación de un TableCellRenderer que, en conjunto con un AbstractTableModel, me permiten mostrar botones (en este caso, los botones para más información de la tabla de medicinas) dentro de una JTable. Particularmente, se tuvo que recurrir a esta técnica porque de no ser así, el *renderer* preestablecido de una JTable solo permite mostrar información utilizando un JLabel. Pero, al sobre-escribir el método del TableCellRenderer y modificarlo para incorporar un JButton y ligarlo al AbstractTableModel (creado también de manera personalizada), se pudo cumplir con los criterios de logros que abarcan aspectos visuales.

```java
public class MyButtonRendererS extends JButton implements TableCellRenderer {

  public MyButtonRendererS() {
    setOpaque(true);
  }

  @Override
  public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {
    setText(value.toString().substring(0,8)); //Oculta indice de label de boton
    return this;
  }
```

Figura 4 - Fragmento de ButtonRenderer con método sobre-escrito

BÚSQUEDA

Una de las implementaciones más complejas, no solo en el aspecto técnico sino también en cuanto al lado creativo, es el sistema de búsqueda implementado en la solución. Dicha idea se concibió durante el proceso de desarrollo del programa y consta en una búsqueda, en este caso de medicinas, que tolere errores en el ingreso de datos y que proponga sugerencias basado en el ingreso. La tolerancia de errores consideraría *typos* de una letra, y corte de la palabra tanto al inicio como al final.

En un primer método, primero se establecen las variables de control y aquellas necesarias para las operaciones. Posteriormente, se almacenan las dos cadenas de caracteres a comparar, con base a su tamaño, para tener control de la mayor y menor. Luego, se establecen contadores que serán utilizados dentro del siguiente ciclo. Un contador de posición es utilizado para recorrer las cadenas y un segundo contador controla un desfasamiento que permitirá comparar en diferentes configuraciones. Un ciclo, luego, cuenta la cantidad de errores carácter por carácter, hasta que encuentra una combinación perfecta o casi perfecta, o el desfasamiento de la palabra menor supere el tamaño de la palabra mayor. Con base a la cantidad de errores se regresa la palabra, precedida con un carácter que identifica si las palabras son iguales, una es un fragmento de otra, son iguales salvo por un *typo*, o si las palabras no se parecen en absoluto.

```java
public static String search (String c, String d){
 String result = "";
 boolean e = false;
 int error = 0;

 String a = d; //menor
 String b = c; //mayor
 if (c.length() <= d.length()){
   a = c; b = d;
 }

 //Contabilización de errores
 int h = 0; //Desplazamiento
 int x = 0; //Posicion

 do {
   error = 0; //limpiar error
   for (x = 0; x < a.length(); x++){
     if (Character.toLowerCase(a.charAt(x)) != Character.toLowerCase(b.charAt(x+h))){
       error++; //diferente
     }
   }
   h++;
 } while ((x+h) != b.length() + 1 && (error != 0 && error != 1)); //Pase sin error
```

Figura 5 - Fragmento de método de búsqueda

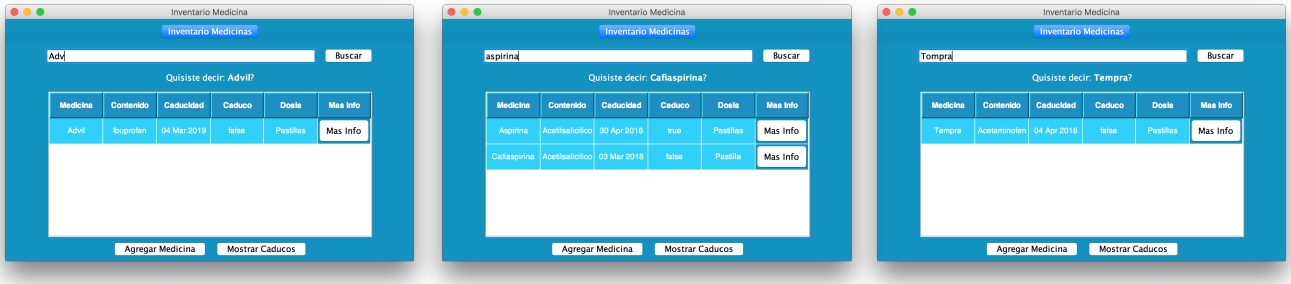

Figura 6 - Demostración de sistema de búsqueda y errores

Esta implementación propia de sistema de búsqueda se realizó para proveer una función más completa a la ya establecida por métodos preestablecidos, que están limitadas a la búsqueda de cadenas de caracteres exactas, y para cumplir con el criterio de logro de búsqueda de medicinas.

VALIDACIÓN

Debido a la posibilidad de ingreso de datos de manera abierta a la hora de crear una medicina, e incluso en algunos casos específicos (como en el ingreso de la fecha), es necesario validar estos datos, para asegurar que cumplan con requisitos preestablecidos.

Tan solo al crear una medicina, se realizan varias validaciones y manejo de errores para evitar una falla en el sistema. Primero que todo, se verifica que para los campos abiertos como Nombre, Contenido, Dosis y Notas, se cumpla con un límite de caracteres, que es esencial debido a los tamaños preestablecidos de los registros a la hora de guardar en el archivo binario. Si el programa encuentra que existe un campo que excede los límites, un mensaje de advertencia le informa al usuario de la situación, y pregunta si sería óptimo editar los datos o guardarlos recortados.

Otra validación verifica que una medicina que esté siendo ingresada, no exista ya. Para ello se realiza una rápida búsqueda dentro de las medicinas ya existentes (utilizando un sistema de búsqueda ya preestablecido, debido a que no hay necesidad de comprobar errores).

En cuanto al ingreso de fechas, se realiza una validación de éstas (es decir, que la fecha sea una válida o existente) utilizando propiedades de la clase Calendar aplicada también a un manejo de excepciones (utilizando bloques try/catch). Durante la ejecución del programa, se crea un fecha Calendar *lentient*, con base a los datos ingresados del usuario. De ser una fecha no válida (como 31 de febrero), el programa arroja una excepción que es atrapada por los bloques mencionados y se toman medidas necesarias, como mostrar un mensaje de advertencia.

```
/////Error #1: Mismo nombre que medicina previa//////////////////////////////////////
if (indice == -1){
  for (int x = 0; x < FrameMainS.getMedList().size(); x++){
    if (FrameMainS.getMedList().get(x).obtenerNombre().equalsIgnoreCase(nombreInput)){
      JOptionPane.showMessageDialog(null, "Dicha medicina ya existe. Por favor elija otro nombre");
      emergency = true;
    }
  }
}
```

```
/////Error #2: Datos largos//////////////////////////////////
if (nombreInput.length() > 20 || contenidoInput.length() > 20 || dosisInput.length() > 10 || notasInput.length() > 250){
  String pregOpt[] = {"Si", "No"};
  int pregunta = JOptionPane.showOptionDialog(null, "Por motivos de memoria los campos de Medicina, Contenido, Dosis y Notas \nestan limitados

  //Cambiar los datos, mandar emergencia
  if (pregunta == 1)
    emergency = true;
}
```

```
/////Error #3: ComboBoxes no fueron seleccionados//////////////////////////////////////

catch (NumberFormatException e){
  JOptionPane.showMessageDialog(null, "Complete los campos de fecha apropiadamente");
  diaCB.setSelectedIndex(0);
  mesCB.setSelectedIndex(0);
  yearCB.setSelectedIndex(0);
  emergency = true;
}
```

```
/////Error #4: Fecha es invalida (ej: feb 31)//////////////////////////////////////

catch (IllegalArgumentException e){ //Thx Java6
  JOptionPane.showMessageDialog(null, "Fecha ingresada no es vailda. Intente de nuevo.");
  diaCB.setSelectedIndex(0);
  mesCB.setSelectedIndex(0);
  yearCB.setSelectedIndex(0);
  emergency = true;
}
```

Figuras 7, 8, 9 y 10 - Fragmentos de validación o manejo de excepciones

COMPARACIONES

```
public static void checarCad2(Calendar ahora, boolean firstTime){
  caducas = new ArrayList<String>();

  for (int x = 0; x < medLst.size(); x++){
    Calendar date = medLst.get(x).obtenerCaducidad();
    if (date.equals(ahora) || date.before(ahora)){
      caducas.add(medLst.get(x).obtenerNombre());
    }
  }
```

Figura 11 - Fragmento con comparación de fechas

Finalmente, se realizan comparaciones tanto en cuanto a fechas, como a nombres de las medicinas. Sin embargo, lo que cabe remarcar con mayor hincapié es la comparación de medicinas caducas, pues es uno de los criterios de logros más esenciales. Después que el usuario ingresa fecha (y ésta se validada como se ha ya mencionado) para obtener las medicinas caducas en dicha fecha, se deben realizar comparaciones entre las medicinas y, específicamente, en cuanto a sus fechas. Para ello, se utilizan ciclos y parte de los métodos de la clase Calendar para comparar distintas fechas.

Palabras: 991

REFERENCIAS

AbstractTableModel (Java Platform SE 7) - Oracle. (s.f.). Recuperado 31 de agosto de 2016 https://docs.oracle.com/javase/7/docs/api/javax/swing/table/AbstractTableModel.html

ArrayList (Java Platform SE 7) - Oracle Help Center. (s.f.). Recuperado 19 de agosto de 2016 https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

Button Table Example : Grid Table « Swing Components - Java. (n.d.). Recuperado 31 de agosto de 2016, de http://www.java2s.com/Code/Java/Swing-Components/ButtonTableExample.htm

Calendar (Java Platform SE 7) - Oracle Help Center. (s.f.). Recuperado 12 de septiembre de 2016, de, from https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html

GridBagConstraints (Java Platform SE 7) - Oracle. (s.f.). Recuperado 3 de septiembre de 2016, de http://docs.oracle.com/javase/7/docs/api/java/awt/GridBagConstraints.html

GridBagLayout (Java Platform SE 7) - Oracle. (s.f.). Recuperado 3 de septiembre de 2016, de https://docs.oracle.com/javase/7/docs/api/java/awt/GridBagLayout.html

JFileChooser (Java Platform SE 7) - Oracle. (s.f.). Recuperado 10 de octubre de 2016, de https://docs.oracle.com/javase/7/docs/api/javax/swing/JFileChooser.html

JTable (Java Platform SE 7) - Oracle Help Center. (s.f.). Recuperado 3 de septiembre de 2016, de https://docs.oracle.com/javase/7/docs/api/javax/swing/JTable.html

SimpleDateFormat (Java Platform SE 6) - Oracle. (s.f.). Recuperado 3 de septiembre de 2016, de http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html