UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

# LABORATORIO 02 - BRAZO ROBÓTICO

# SEGUNDA ENTREGA

Peña Bolívar - 20181020146, Preciado Ballen - 20182020122, Suárez Sánchez - 20182020107

***Resumen*—En este informe veremos cómo funciona un brazo robótico, haciendo uso de la programación orientada por objetos POO, el meta patrón Modelo Vista Controlador MVC, y la programación en tres capas Lógica, Presentación, Persistencia, además veremos cómo el uso de los modelamientos de los Diagramas De Clases y Diagramas Caso De Uso, nos ayudarán a entender cómo funciona el brazo robótico.***

## I. INTRODUCCIÓN

Un brazo robótico es un sistema que estará compuesto por varias partes; una base, cuatro articulaciones, tres secciones y una garra que abre y cierra. El sistema brazo robótico en este documento será uno que tiene ciertas características y nos representa algunas más.

La base tendrá una altura de 3 cm, estará siempre fija en un eje, será en este caso el eje X, a su vez la base será móvil horizontalmente sobre el eje, tendrá un mínimo valor en cero y un máximo en ochocientos, además la base tendrá un ángulo de rotación que estará entre los 0° y los 360°, esto para que durante la ejecución podamos modificar ángulos de las articulaciones y poder calcular coordenadas de posición final de las secciones.

Las articulaciones son uniones entre ya sea base - sección, o seccion - seccion. las articulaciones tomarán una parte en cada sección, para cada una son 3 cm. Las articulaciones son las partes fundamentales del sistema brazo robótico, son la principal fuente del movimiento de cada sección y el brazo robótico en su totalidad.

Las secciones son por llamarlos de alguna manera parales que permiten que el brazo se pueda mover de cualquier manera siguiendo las restricciones; cada sección tendrá un ángulo de apertura que estará entre los 5° y los 175°.
La garra es la parte que podrá abrirse o cerrarse, estará ubicada en la articulación cuatro que unirá la última sección con la misma.

## II. DESARROLLO DE CONTENIDO

### PRIMERA ENTREGA

### DIAGRAMA DE CLASES

Por medio de este diagrama nos planteamos el modelamiento de nuestra aplicación considerando desde su arquitectura hasta los objetos, características y métodos que conformaría cada clase, la aplicación se desarrollo implementando el Meta-patrón de diseño MVC (Modelo-Vista-Controlador), es decir, en este diagrama describimos detalladamente lo que debía estar presente en el sistema basados en este tipo de arquitectura, además nos permitió comprender la visión general de los esquemas de la aplicación basados en los requerimientos dados por el docente.

Basados en lo enseñado en clase, teníamos clara la siguiente estructura:

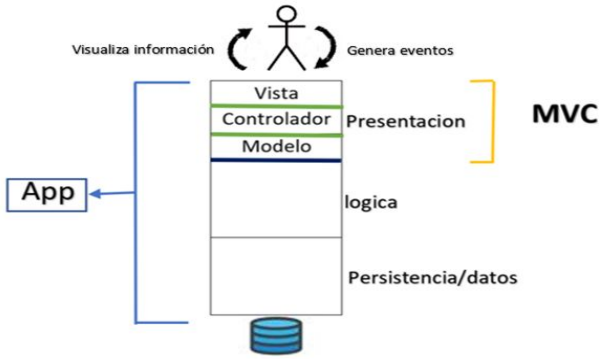



**Imagen 1.** Estructura MVC.(Tomado de: clase de Programación Avanzada. Universidad Distrital. Elaboración de la propuesta Intelectual: Oswaldo Romero). [1]

Ya teniendo clara la estructura se nos enseñó de igual manera como esto se veía en el diagrama de clases a nivel general y las

separaciones presentadas en la **Imagen 1**, esto fue de vital importancia para la construcción de nuestra aplicación, y lo representamos de la siguiente forma:

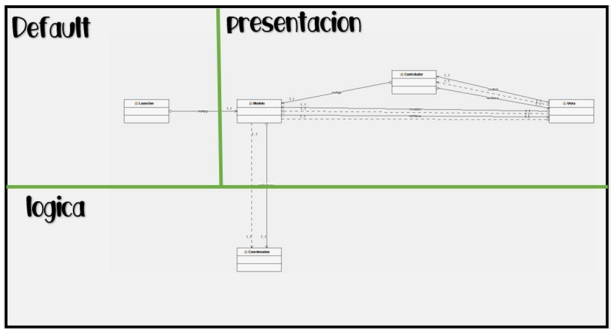
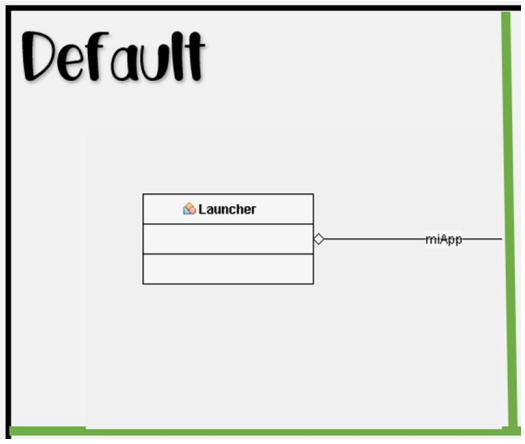


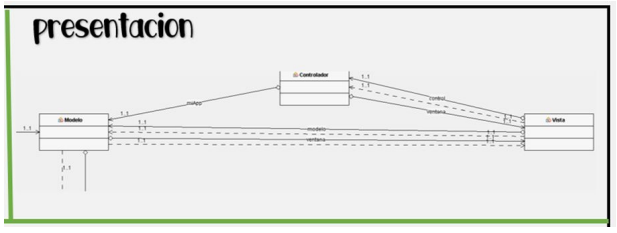

**Imagen 2.** Diagrama con solo nombre de las clases dentro del croquis de MVC. (Elaborado por: Autores. Idea intelectual del planteamiento: Oswaldo Romero. Universidad Distrital. Programación Avanzada). [1]

**VISTA AMPLIADA IMAGEN 2**

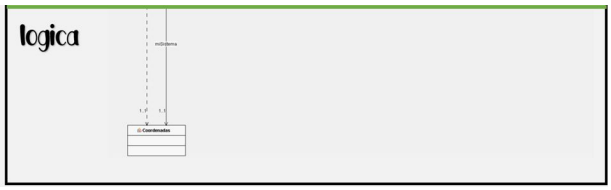
**Default:**



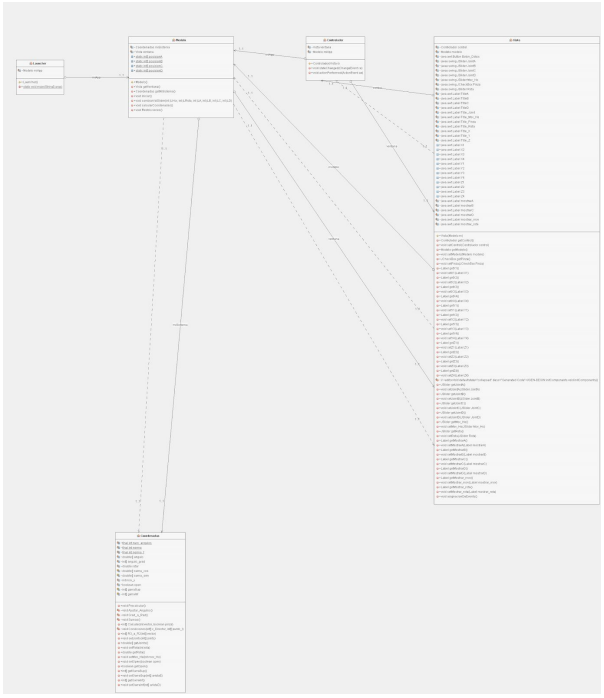

**Presentación:**



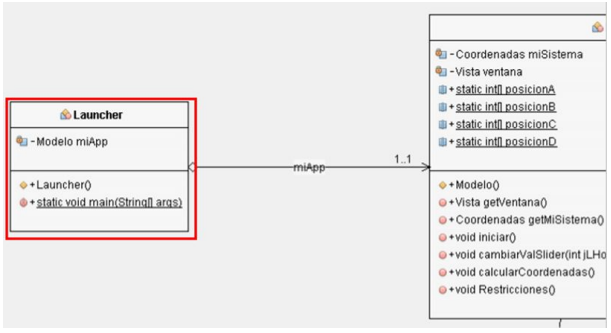

**Lógica:**



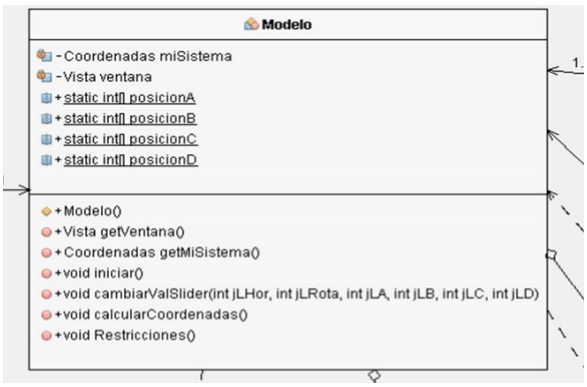

A continuación se presenta el diagrama de clases ya desplegado y detallado:



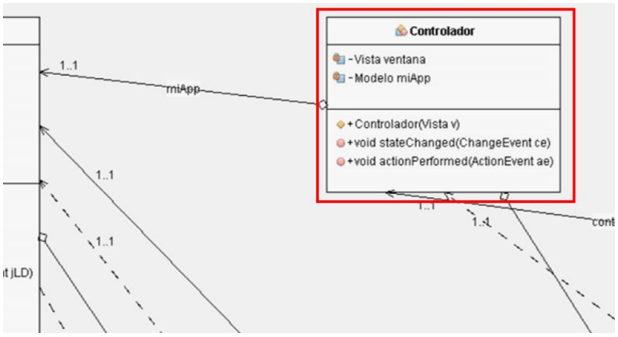
**Clase Launcher:**



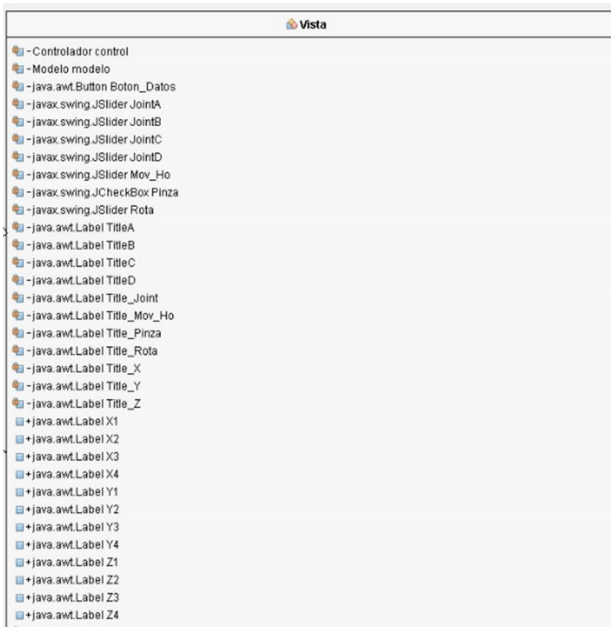

**Clase Modelo:**




**Clase Controlador:**




**Clase Vista:**





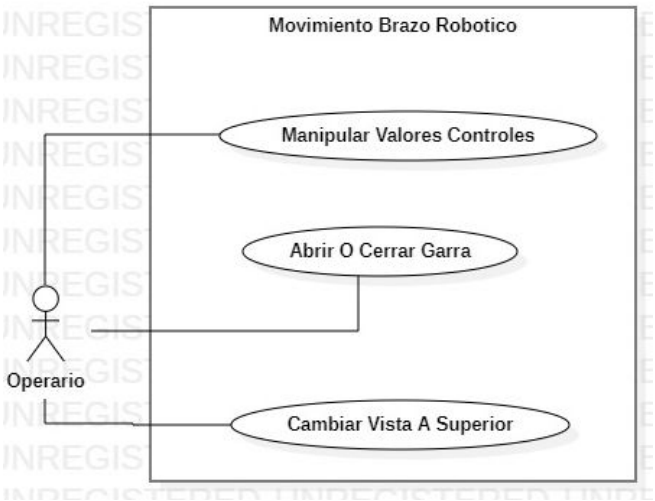

**Clase Coordenadas:**

**DIAGRAMA CASO DE USO**

## 1. DESCRIPCIÓN CASO DE USO.

| | |
|---|---|
| CASO DE USO | MOVIMIENTO BRAZO ROBÓTICO |
| ACTORES | Operario |
| DESCRIPCIÓN | Una vez los valores de los controles del brazo robótico sean modificados, el sistema retorna el posicionamiento seccional en un sistema de coordenadas R3 |
| PRECONDICIÓN | El sistema estará en sus valores predeterminados al inicio de la interacción con el operario, es decir los valores de inicio de ejecución serán los mismos que tendrá el sistema siempre al ejecutarse. |
| POST - CONDICIÓN | El operario al finalizar la ejecución obtendrá un registro en coordenadas R3, las cuales significaron la posición en la que ha quedado cada sección del brazo robótico. |
| ACTOR PRIMARIO | Operario |
| FLUJO PRINCIPAL | ACCIONES |
| | 1. El sistema tendrá todos los valores de sus controles en estado predeterminado, es decir los valores de las articulaciones, movimiento horizontal y rotacional serán los establecidos por el programador. |
| | 2. El operario manipulara los valores de los controles: articulaciones, movimiento horizontal, y rotacional. |
| | 3. El sistema se encargará de que el usuario tenga visible los valores en los que se encuentran los controles. |
| | 4. El usuario acciona el evento aceptar, el cual ejecuta una serie de métodos que calculan los valores de posicionamiento en coordenadas R3. |
| | 5. El sistema retorna al usuario en coordenadas R3 los valores de posicionamiento de cada sección del brazo. |
| EXCEPCIONES | ACCIONES |
| | Si el operario cambia los controles de tal manera que genere un choque con uno de los muros se generarán problemas. |
| | E.1 - El sistema informa al operario por medio de un panel que está excediendo los límites, se podrá seguir con la ejecución pero estará el aviso. |

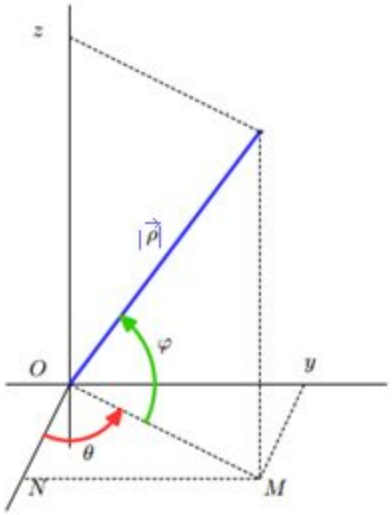
## 2. DIAGRAMA CASO DE USO.




## 3. TEORÍA IMPLEMENTADA - CÁLCULOS.

**Cálculo de coordenadas:**
Dado que estamos en el espacio, se utilizan las coordenadas esféricas las cuales son:

$$x=p*\cos(\varphi)\cos(\theta)$$
$$y=p*\cos(\varphi)\sen(\theta)$$
$$z=p*\sen(\varphi)$$




Luego por las condiciones que fueron indicadas previamente:

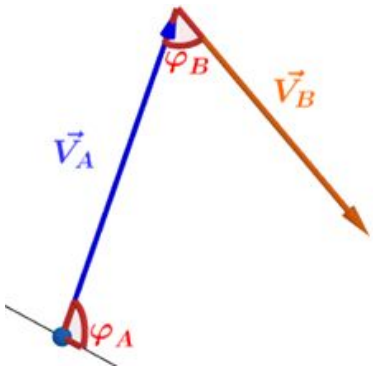
**p=144**

Entonces la ecuación del primer vector es:

**V={144 cos(φ) cos(θ),144 cos(φ)sen(θ),144sen(φ)}**

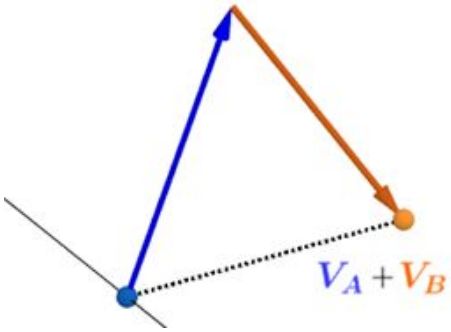
Puesto que tenemos cuatro ángulos para cada sección asignaremos un subíndice con el fin de identificarlos:

$$\varphi_A, \varphi_B, \varphi_C, \varphi_D$$




Ahora dado que el primer vector $(V_A)$ es la base de donde parte el segundo vector $(V_B)$ se realiza una suma para saber la ubicación del vector $(V_B)$:

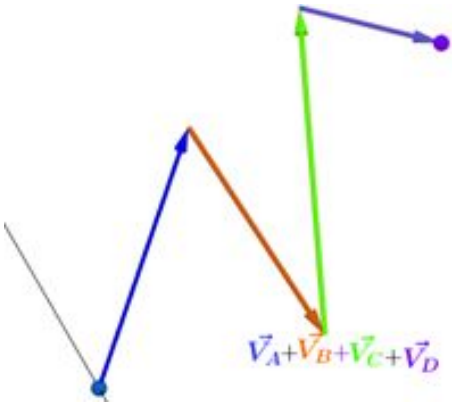
$$V_B = \{144cos\theta[cos\varphi_A + cos\varphi_B], \\ 144cos\theta[sen\varphi_A + sen\varphi_B], \\ 144[sen\varphi_A + sen\varphi_B]\}$$




Finalmente se aplica la misma lógica para todas las secciones restantes:

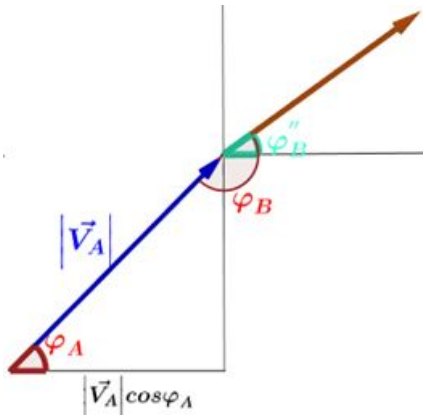
$$V_C = \{144cos\theta[cos\varphi_A + cos\varphi_B + cos\varphi_C], \\ 144cos\theta[sen\varphi_A + sen\varphi_B + sen\varphi_C], \\ 144[sen\varphi_A + sen\varphi_B + sen\varphi_C]\}$$

$$V_D = \{144cos\theta[cos\varphi_A + cos\varphi_B + cos\varphi_C + cos\varphi_D], \\ 144cos\theta[sen\varphi_A + sen\varphi_B + sen\varphi_C + sen\varphi_D], \\ 144[sen\varphi_A + sen\varphi_B + sen\varphi_C + sen\varphi_D]\}$$




Pero debido a que el origen del vector $V_B$ está sobre el vector $V_A$ hay que realizar un ajuste en el ángulo $\varphi_B$ con el fin de que $V_B$ quede sobre el origen, y se cumpla la ecuación anterior:

$$\varphi_B'' = \varphi_B - 90 - cos^{-1}(sen\varphi_A)$$



## SEGUNDA ENTREGA

### CONFIGURACIÓN TEÓRICA DEL BRAZO ROBÓTICO

A continuación se muestra el planteamiento gráfico teórico del brazo robótico, que en la parte de **aplicación** se aplicará al desarrollo realizado :

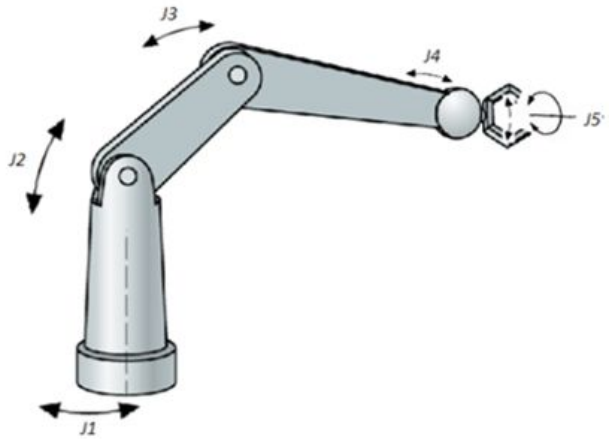



**Imagen 4.** Esquema del diseño del brazo articulado. (Tomado de: https://core.ac.uk/reader/132529762). [2]

### MVC

Para esta segunda entrega, el MVC en su estructura general de clases queda como se muestra a continuación:

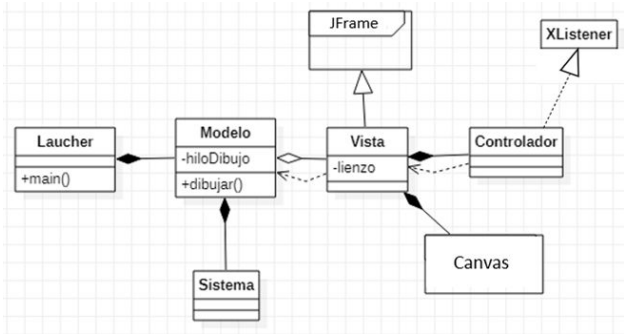
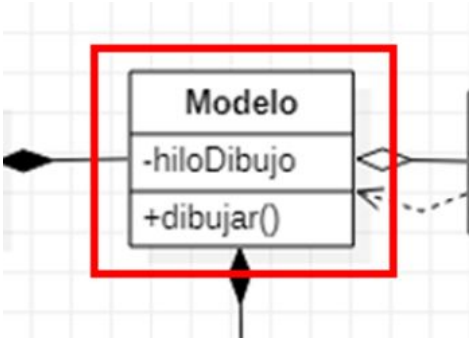
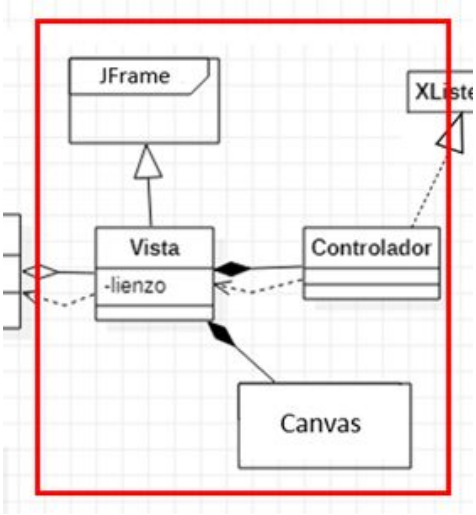



**Imagen 3.** Estructura MVC - Segunda entrega.(Tomado de: clase de Programación Avanzada. Universidad Distrital. Elaboración de la propuesta Intelectual: Oswaldo Romero). [1]

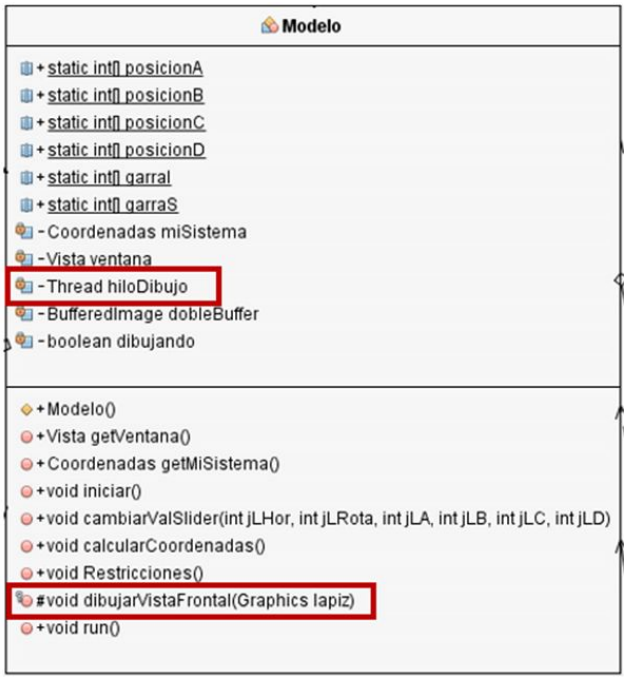
**Clase Modelo:**




**Clase Vista:**



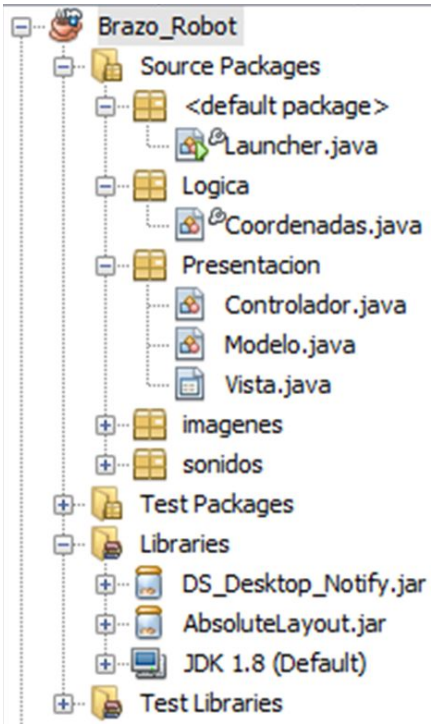

**Clases tomadas del Diagrama de Clases de la Segunda Entrega:**



**Clase Vista:**


**ESTRUCTURA DEL PROYECTO**

La estructura del proyecto presentada para esta segunda entrega tiene un cambio en la carpeta **Libraries,** donde se añadió la librería "DS Notify"( que en el apartado "**Librería DS Notify**" encontrará una descripción de cómo es utilizada en el desarrollo),de acuerdo a esto la estructura es:




**IMPLEMENTACIÓN EN EL CÓDIGO**

**Clase Modelo:**

```
15  public class Modelo implements Runnable{
16
17      public static int[] posicionA= new int[3];
18      public static int[] posicionB= new int[3];
19      public static int[] posicionC= new int[3];
20      public static int[] posicionD= new int[3];
21      public static int[] garraI= new int[3];
22      public static int[] garraS= new int[3];
23
24
25
26      private Coordenadas miSistema = new Coordenadas();
27      private Vista ventana;
28      private Thread hiloDibujo;
29      private BufferedImage dobleBuffer;
30      private boolean dibujando;
31
32      // Métododos generados para ocultación de informació
33      public Modelo() {
34          dibujando=false;
35      }
```

```
public class Modelo implements Runnable{
```

Implementación de la interface Runnable y declara la función run(), que se encarga de definir las clases que implementen esta interfaz. Más adelante se mostrará la definición de la función run.

```
private Thread hiloDibujo;
```

Creación del hilo: hiloDibujo de tipo Thread..

```
private BufferedImage dobleBuffer;
```

Esto lo implementamos para corregir efectos de la animación, lo que hace es que no nos va a dibujar sobre la ventana directamente sino en buffer intermedio que se refiere a un contexto gráfico en memoria, cuando se actualiza el el brazo se hace por medio de una transferencia, que se vaya lo dibujado desde el contexto en memoria a la ventana, luego se dibuja de nuevo en el contexto gráfico de memoria y se pasa a la ventana de manera simple y muy rápida

```
26      private Coordenadas miSistema = new Coordenadas();
27      private Vista ventana;
28      private Thread hiloDibujo;
29      private BufferedImage dobleBuffer;
30      private boolean dibujando;
31
32      // Métododos generados para ocultación de informació
33      public Modelo() {
34          dibujando=false;
```

dibujando se volver true cuando se cree el subproceso del hilo y permanecerá así, dibujando, mientras permanezca así como true, esto se especificara por medio de un while en la función run.

```
if(hiloDibujo == null){
 hiloDibujo = new Thread(this);
 dibujando = true;
 hiloDibujo.start();
```

Creamos un subproceso, es decir, el objeto de la clase Thread, desde este objeto se llama la función start para poner en ejecución el subproceso.

```
protected void dibujarVistaFrontal(Graphics lapiz) {

    lapiz.setColor(Color.CYAN);
    lapiz.fillRect(0, 0, getVentana().getjSPVistaFrontal().getWidth(), getVentana().getjSPVistaFrontal().getHeight());
    lapiz.setColor(new Color(65,34,39));
```

Realizamos la utilización de Graphics, el contexto Graphics permite pintar la pantalla, el objeto Graphics controla el contexto de los gráficos.

Se emplea la clase Color, que es lo que nos permite definir los métodos y las constantes para manipulación de los colore, como ya sabemos estos se crean en base al esquema rojo/verde/azul (RGB).

| | | |
|---|---|---|
| `public final static Color CYAN` | cyan | 0, 255, 255 |

El método fillRect crea nuestro lienzo que es un rectángulo relleno en el color definido.

Puntos de drawLine:

```
public void drawLine( int x1, int y1, int x2, int y2 )
```

Ejecución de run:

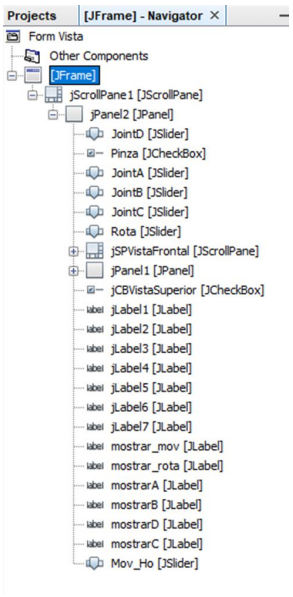
```
@Override
public void run() {
    Canvas lienzo = getVentana().getVistaFrontal();
    dobleBuffer = new BufferedImage(getVentana().getVistaFrontal().getWidth(), getVentana().getVistaFrontal().getHeight(), Buf
    Graphics lapiz = dobleBuffer.getGraphics();  // Obtener el contexto gráfico
    Graphics lapizLienzo = lienzo.getGraphics();
    System.out.println("Hilo ejecutando");

    while (dibujando) {
        dibujarVistaFrontal(lapiz);
        lapizLienzo.drawImage(dobleBuffer, 0, 0, lienzo);

    }
}
```

**Clase Vista:**

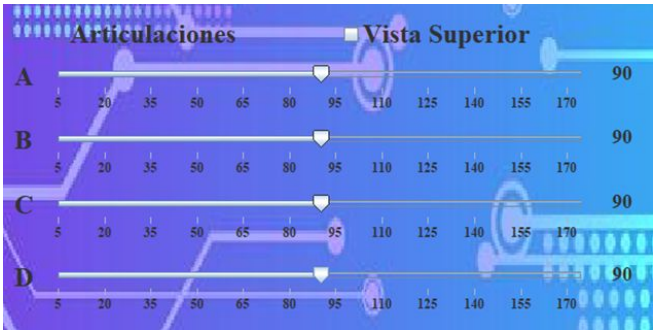
Estructura de “Form Vista”, vista Navegador:



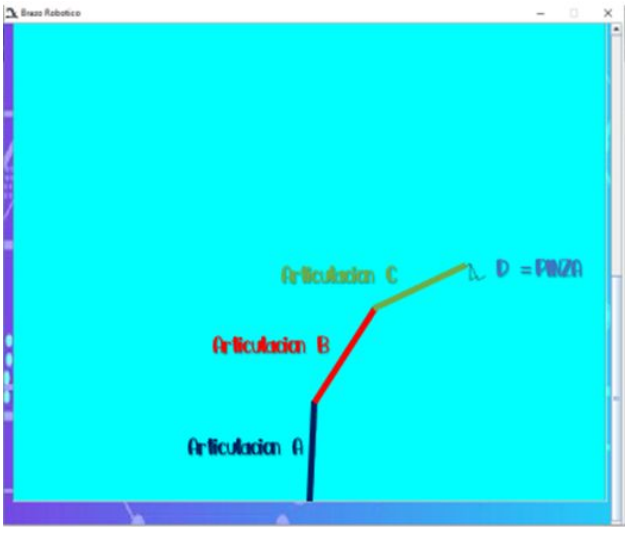

**APLICACIÓN**

Articulaciones en interfaz gráfica:

Controles:

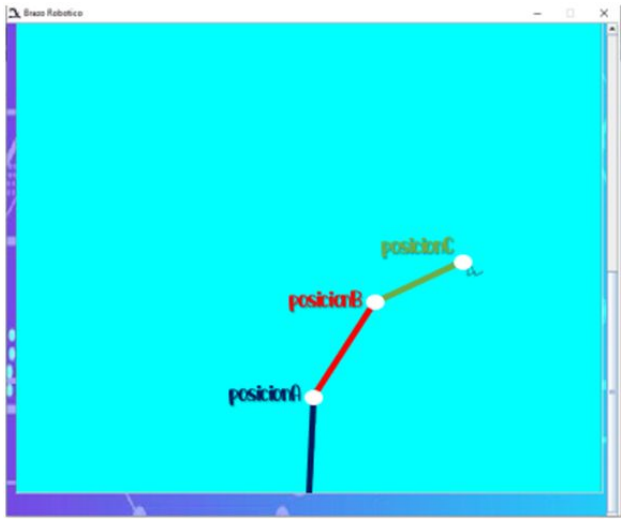



Secciones en el brazo:



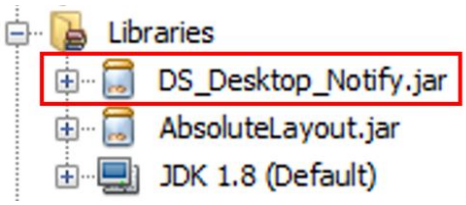
Coordenadas en la Lógica de la aplicación:

Secciones:

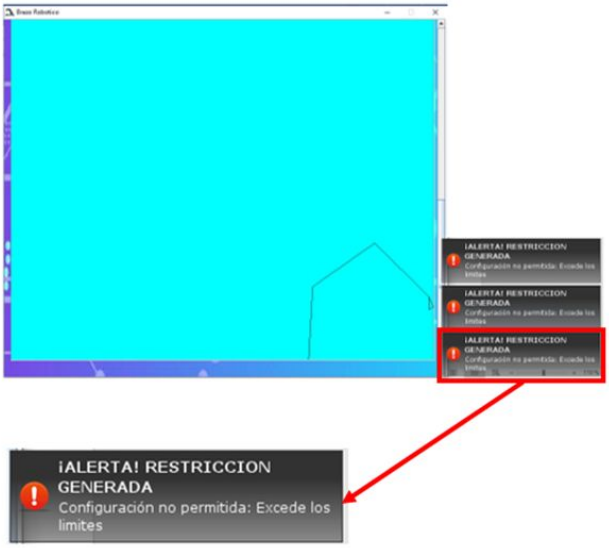



Pinza:


**LIBRERÍA "DS NOTIFY"**

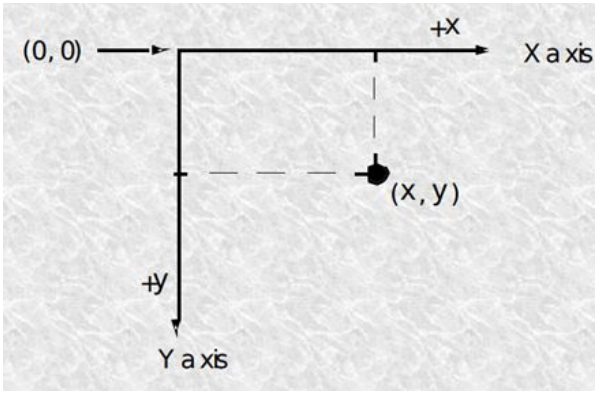



Esta librería es necesaria para ejecutar la aplicación, fue implementada con el fin de brindar al usuario una mejor experiencia en el uso de la misma, haciendo que cuando el usuario realice una configuración que exceda los límites permitidos se genere una notificación que da la sensación de que la notificación viene del equipo donde se está ejecutando, el número de restricciones emergentes es igual al número de restricciones que está infringiendo, estas se desaparecen a los dos segundos.




**ANEXOS**

**Sistema de Coordenadas de java, unidad de medida: pixeles.**




**Arreglos y ejes:**

**X** cuando está en la posición **[0]**
**Y** cuando está en la posición **[1]**
**Z** cuando está en la posición **[2]**

## III. CONCLUSIONES

Con la realización de este laboratorio pudimos poner en práctica todos los conocimientos adquiridos en la parte teórica de la clase, ejemplo de esto fue la implementación de la arquitectura en tres capas, donde se utiliza la lógica para la parte metódica y cálculos de la aplicación; presentación, donde irá implementado el MVC, y finalmente; la persistencia que no fue usada debido a que no se utilizó estructuras de datos externos como lo son las bases de datos. También la implementación del meta-patrón de diseño MVC, pusimos en

práctica lo aprendido referente a UML específicamente en el diagrama de clases y la construcción y redacción de los casos de uso. Además de esto al iniciar el proceso de programar esta aplicación notamos cambios en la manera de programar, gracias a la aplicación de Modelo-Vista-Controlador apreciamos una codificación entendible, organizada, clara y que buscamos orientar hacia el cumplimiento de los principios SOLID en el sentido de que esta aplicación permitiese en lo posible llegar a ser flexible, con alta cohesión y bajo acoplamiento.

Por medio de esta segunda entrega logramos poner en práctica lo aprendido de Threads, Canvas, Graphics, continuando con la aplicación de Modelo-Vista-Controlador, los conocimientos y conceptos para desarrollar las aplicaciones multihilo se trataron de manera específica comprendiendo a través de la puesta en marcha del hilo en la aplicación que la creación de un nuevo hilo se va a constituir como una característica que permite a la aplicación realice varias tareas a la vez, y que en esta ocasión lo utilizamos para dibujar el brazo robótico en el canvas, continuamos con el desarrollo de una aplicación flexible, con alta cohesión y bajo acoplamiento que se evidenció al momento de implementar lo que se pedía en este segundo laboratorio fue de manera precisa, flexible y con ausencia de rigidez. Con el uso de hilos buscamos maximizar los recursos del ordenador, simplificando el diseño de la aplicación, y aprendiendo el manejo de cada parte de Graphics y funciones tales como el doublebuffer.

## IV. REFERENCIAS

[1]Clase de Programación Avanzada. Universidad Distrital Francisco José de Caldas. Profesor: Oswaldo Romero.
[2]Ingeniería Electrónica Industrial y Automática. Disponible en: https://core.ac.uk/reader/132529762