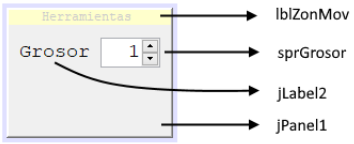
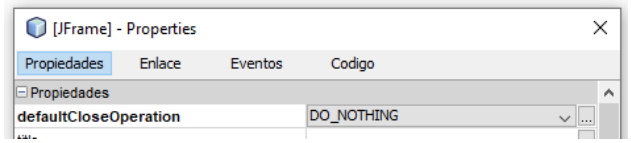
- Proyecto que dibuje primitivas geométricas (líneas) mediante el Mouse (Ejemplo 04):
    - Se creará el proyecto jOpenGLEjem4
    - Se agregarán las librerías GLFW y OpenGL
    - Programando la ventana de Herramientas (*clsVenDeHerr.java*):
        - Se va a interactuar con una ventana JFrame que contenga mediante un Spinner, el Grosor de línea que se dibujará con GLFW:



        - Esta ventana deberá tener restricciones en su uso, los cuales se configuran desde su panel "Propiedades"; tales como:

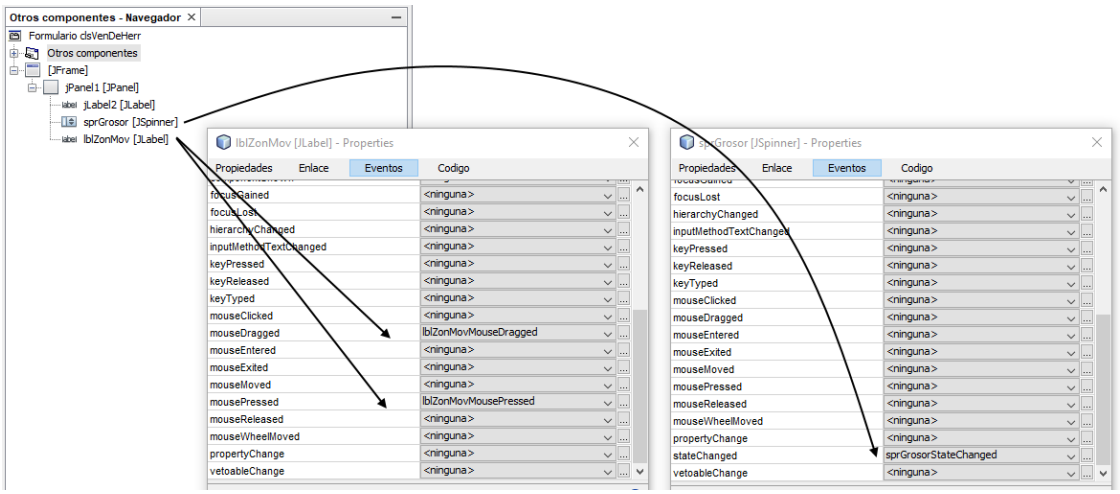
          Desactivación de cerrado de ventana (icono "X" o Ctrl+F4)



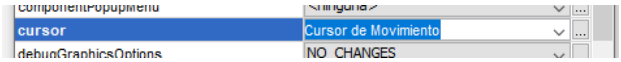
          Retirado de la Barra de título y sus íconos de minimizar, maximizar y cerrar, activando a:

          Desactivando el modo de redimensión de la ventana:

          Además, dicha ventana debe contener un contenedor (JPanel), 2 controles de tipo *Etiqueta* (JLabel) y una caja mediante indicadores de incremento o decremento (JSpinner), los cuales deberán configurarse como:



De la misma forma debe configurarse la propiedad de la etiqueta *lblZonMov* para cambiar el ícono del mouse:



Código que deben contener los eventos indicados anteriormente, asimismo declarar las variables públicas de la clase:

```
12  public class clsVenDeHerr extends javax.swing.JFrame {
13
14      /**
15       * Creates new form clsVenDeHerr
16       */
17      int posVenX, posVenY;
18      float valGrosor;
19
20      public clsVenDeHerr() {
21          initComponents();
23          setLocation(275, 155); //pos. temporal... !!!mejorar de forma dinámica
24      }
25      /** This method is called from within the constructor to initialize the form ...5 lines */
30      @SuppressWarnings("unchecked")
31      Generated Code
112
113     private void lblZonMovMousePressed(java.awt.event.MouseEvent evt) {
114         //Se obtiene la coord. x;y al presionar botones del mouse
115         posVenX = evt.getX();
116         posVenY = evt.getY();
117     }
118
119     private void lblZonMovMouseDragged(java.awt.event.MouseEvent evt) {
120         //Se ejecuta el desplazamiento de la Ventana según las coord. del mouse
121         this.setLocation(this.getLocation().x + evt.getX() - posVenX, this.getLocation().y + evt.getY() - posVenY);
122     }
123
124     private void sprGrosorStateChanged(javax.swing.event.ChangeEvent evt) {
125         //Se obtiene el valor de tipo entero a tipo Float en la variable valGrosor (grosor de línea)
126         valGrosor = ((Integer)sprGrosor.getValue()).floatValue();
127     }
```

Los eventos *MousePressed* y *MouseDragged*, permitirán mover la ventana siempre y cuando se haya hecho clic y se arrastre desde la etiqueta (lblZonMov), esto es necesario debido a que se activo la casilla "***Undecored***" (ventana sin barra de titulo y gestión de esta) impidiendo esta función propia de la ventana JFrame.

Con respecto a la variable *valGrosor*, esta capturará el valor que tenga *sprGrosor*; dicha variable pública ayudará a configurar el grosor de línea cuando se dibuje una línea en la ventana de dibujo.

- Programando la aplicación General

  - Se creará una clase Java que gestione la interacción de la ventana – Herramientas con una ventana OpenGL mediante GLFW, el manejo de sombreadores y la ejecución del programa Sombreador; asimismo, el conversor de coordenadas del mouse con los vértices GLFW; dicha clase se llamará *clsGlfwConSomb.java*:

```
package jopenglejem4;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.FloatBuffer;
import java.util.List;
import org.lwjgl.*;
import org.lwjgl.glfw.*;
import org.lwjgl.opengl.*;

import static org.lwjgl.glfw.GLFW.*;
import static org.lwjgl.opengl.GL30.*;
import static org.lwjgl.system.MemoryUtil.*;

/**
 *
 * @author PC-MAX
 */
public class clsGlfwConSomb {
    private static int ID;
    private static long ventana;
    private static clsVenDeHerr objVDH;

    //Constructor
    public clsGlfwConSomb(clsVenDeHerr ven1){
        objVDH = ven1; //obtener la ventana clsVenDeHerr para gestionar sus métodos
    }

    private static vcd ini_GLFW(long ven, int ancho, int alto, String titulo){
        vcd objVCD = new vcd();

        glfwInit();
        glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
        glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
        glfwWindowHint(GLFW_RESIZABLE, GLFW_FALSE);

        ven = glfwCreateWindow(ancho, alto, titulo, NULL, NULL);
        if(ven == NULL){
            objVCD.setMsj("Fallo al crear la ventana con GLFW");
            objVCD.setVentana(NULL);
            return objVCD;
        }

        glfwSetFramebufferSizeCallback(ven, (objVentana, an, al)->{
            glViewport(0, 0, an, al);
        });

        glfwSetKeyCallback(ven, (objVentana, tecla, scancode, accion, mods) -> {
            if (tecla == GLFW_KEY_ESCAPE && accion == GLFW_RELEASE) {
                glfwSetWindowShouldClose(objVentana, true);
                objVDH.dispose(); //cerrar clase venDeHerr
            }
        });

        GLFWVidMode vidmode = glfwGetVideoMode(glfwGetPrimaryMonitor());
        glfwSetWindowPos(ven,(vidmode.width() - ancho) / 2,(vidmode.height() - alto) / 2);

        glfwMakeContextCurrent(ven);
        glfwSwapInterval(1);

        GL.createCapabilities();

        objVCD.setVentana(ven);
        return objVCD;
    }
```

El evento ini_GLFW, ya fue explicado en ejercicios anteriores, por lo que; la modificación que tendrá, yacerá en su *constructor* pasando la clase *clsVenDeHerr* a un objeto de nombre *objVDH* del cual se rescatarás sus variables públicas tales como *valGrosor.*

- Con respecto a los eventos *Sombreador* y *verifErrorComp*, estos ya fueron explicados por lo que no tienen cambio alguno.

- El evento *genObjADibMod* tiene la misma estructura que el del *genObjADib* (ejercicios anteriores) solo que, en este se hacen las modificaciones de ingreso de Arreglo de listas: *alVertices*, *alColores* que contendrán los vértices asignados por el mouse para el dibujo de líneas (evento que se realiza en la clase "Main"):

```
154     public int genObjADibMod(List<Float> alVertices,List<Float> alColores){
            int pVBO = 0,cVBO = 0 , VAO = 0;
156         VAO = glGenVertexArrays();
157         glBindVertexArray(VAO);
158         int i;
159
160         //Convertir píxeles en Coord basadas en un Vértice
161         float[] vertices = new float[alVertices.size()];
162         i = 0;
163         for (Float f : alVertices) vertices[i++] = (f != null ? f : Float.NaN); // Evitar el NullPointerException
164         float[] colores = new float[alColores.size()];
165         i = 0;
166         for (Float f : alColores) colores[i++] = (f != null ? f : Float.NaN); // Evitar el NullPointerException
167
168         //Atributos para posición de coordenadas 3D (vértice)
169         pVBO = glGenBuffers();
170         glBindBuffer(GL_ARRAY_BUFFER, pVBO);
171         FloatBuffer vbo = BufferUtils.createFloatBuffer(vertices.length); //vertex buffer object
172         vbo.put(vertices);
173         vbo.flip();
174         glBufferData(GL_ARRAY_BUFFER, vbo, GL_STATIC_DRAW);
175         glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
176         glEnableVertexAttribArray(0);
177         glBindBuffer(GL_ARRAY_BUFFER, 0); //liberando búfer vinculado
178
179         //Atributo para posición de colores RGB
180         cVBO = glGenBuffers();
181         glBindBuffer(GL_ARRAY_BUFFER, cVBO);
182         FloatBuffer cbo = BufferUtils.createFloatBuffer(colores.length); //color buffer object
183         cbo.put(colores);
184         cbo.flip();
185         glBufferData(GL_ARRAY_BUFFER, cbo, GL_STATIC_DRAW);
186         glVertexAttribPointer(1, 3, GL_FLOAT, false, 0, 0);
187         glEnableVertexAttribArray(1);
188         glBindBuffer(GL_ARRAY_BUFFER, 0); //liberando búfer vinculado
189         glBindVertexArray(0); //Liberando array
190         glLineWidth(objVDH.valGrosor);//grosor de línea
191         return VAO;
192     }
```

El objetivo de usar el *arreglo de listas* es que, no se sabe cuantos vértices se asignarán con el mouse, es por ello que se usa la *forma dinámica de esta estructura*; para posteriormente, mediante repetidores; asignar los datos de este *arreglo de listas* en Arreglos propiamente dichos como: *vertices* y *colores*, por lo demás el código siguiente a este, ya fue explicado.

- Para el evento *ini_ventana*, se ejecuta la ventana de dibujo creada, gestionando cualquier tipo de alerta, como también se integran los sombreadores; para el evento *use(…)* se hacen modificaciones ya que estos traerán el objecto que genera los puntos (*obj*), el tamaño del arreglo de listas de los vértices(*tamA*) y la ventana de dibujo(*v*); con estos datos se vinculan los vértices a dibujar se especifica que tipo de dibujo realizar (GL_LINE_STRIP) y se activa el doble búfer:

```
194     public long ini_ventana(int ancho, int alto, String titulo){
195         //Se Inicializa GLFW, si todo es correcto; se crea la ventana
            vcd venConDat = new vcd();
197         venConDat = ini_GLFW(ventana,ancho, alto, titulo);
198         if (!"".equals(venConDat.getMsj())) {
199             System.out.println(venConDat.getMsj());
200             glfwTerminate();
201             return NULL;
202         }
203         else {
204             ventana = venConDat.getVentana();
205             Sombreador("verticeSomb.vs","fragmentoSomb.fs",false);
206         }
207         return ventana;
208     }
209
210     public void use(int obj,int tamA,long v){
211         glUseProgram(ID);
212         glBindVertexArray(obj); //vincular el objeto a dibujar con su Arreglo de vértices
213         glDrawArrays(GL_LINE_STRIP, 0,tamA/3);//uso de puntos (vértices basados en 3 coord)
214         glfwSwapBuffers(v); //activar el doble búfer
215
216     }
217
218     public float cPAV(double pos,int med,boolean eje){//(c)onvertir (P)ixeles (A) coord de tipo (V)értice
219         float con;
220         if(eje){
221             con = (float)((2*pos)/med-1); //fórmula matemática para el eje x
222         }else{
223             con = (float)(1-(2*pos)/med); //fórmula matemática para el eje y
224         }
225         return con;
226     }
227 }
```

Para el evento *cPAV*; este se encargará de convertir las coord. generadas por el mouse en vértices de dibujo (esto debido a que GLFW está configurado como coord. con puntos céntricos del tipo de dato float en el rango de -1.0 a 1.0)

- Al ejecutar el código principal desde el "Main", se instanciará la ventana de herramientas: *objVDH*; se declarará el tamaño fijo de la ventana principal (*_Ancho*, *_Alto*); se instanciará la clase de los sombreadores en *objGCS*, como también la instancia del gestor de coord. del mouse y eventos de dibujo con este: *objCM*; se crearán los arreglos de listas: *alVertices*, *alColores* y se asignara la variable *objDib* que contenga los vértices a dibujar:

```
package jopenglejem4;
import java.util.ArrayList;
import java.util.List;
import static org.lwjgl.glfw.Callbacks.*;
import static org.lwjgl.glfw.GLFW.*;
import static org.lwjgl.opengl.GL30.*;
import static org.lwjgl.system.MemoryUtil.*;


/**
 *
 * @author PC-MAX
 */
public class JOpenGLEjem4 {
    /**
     * @param args the command line arguments
     */

    public static void main(String[] args) {

        clsVenDeHerr objVDH = new clsVenDeHerr();

        objVDH.setVisible(true);

        final int _Ancho = 1024;
        final int _Alto = 720;
        long ventana = NULL;
        clsGlfwConSomb objGCS = new clsGlfwConSomb(objVDH);
        clsCoordMouse objCM = new clsCoordMouse(); //Forma de recuperar coordenadas del Mouse
        List<Float> alVertices = new ArrayList<>();
        List<Float> alColores = new ArrayList<>();
        int objDib, i;
        try {
            ventana = objGCS.ini_ventana(_Ancho, _Alto, "Ejem 4: OpenGL con Java - Primitivas Con Mouse");
            if(ventana != NULL){

                //Se genera el dibujo con el arreglo de puntos
                objDib = objGCS.genObjADibMod(alVertices,alColores);

                glfwSetCursorPosCallback(ventana, (long objVentana, double xPos, double yPos)->{
                    /*xPos y yPos son variables enfrascadas en glfwSetCursorPosCallback, por lo que en esta
                     sección sería difícil enviar los valores de xPos y yPos hacia variables locales del Main
                    declarando la clase en el Main que encapsule a xPos y yPos y devuelva sus valores.
                    */
                    if(!alVertices.isEmpty() && objCM.getEsPosIni() != 1){
                        alVertices.remove(alVertices.size()-1); //borra coord x
                        alVertices.remove(alVertices.size()-1); //borra coord y
                        alVertices.remove(alVertices.size()-1); //borra coord z

                        //Borrando color de punto
                        alColores.remove(alColores.size()-1); //borra color R
                        alColores.remove(alColores.size()-1); //borra color G
                        alColores.remove(alColores.size()-1); //borra color B
                    }else{
                        objCM.setEsPosIni(0);
                    }
                    alVertices.add(objGCS.cPAV(xPos,_Ancho,true));
                    alVertices.add(objGCS.cPAV(yPos,_Alto,false));
                    alVertices.add(0.0f); //pos Z no utilizada pero obligatoriamente declarada

                    //Agregando color al punto
                    alColores.add(0.0f); //R
                    alColores.add(0.0f); //G
                    alColores.add(1.0f); //B
                    objCM.setX(xPos);
                    objCM.setY(yPos);
                });

                glfwSetMouseButtonCallback(ventana,(long objVentana, int boton, int accion, int mod)->{
                    if(boton == GLFW_MOUSE_BUTTON_LEFT && accion == GLFW_RELEASE){
                        objCM.setEsPosIni(1);
                    }
                });

                //Bucle de renderizado
                while (!glfwWindowShouldClose(ventana)) {
                    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);//color de fondo - Blanco
                    glClear(GL_COLOR_BUFFER_BIT); // limpiar el FrameBuffer

                    //Se genera el dibujo con la lista de arreglos de puntos según el movimiento del mouse
                    objDib = objGCS.genObjADibMod(alVertices,alColores);

                    //Se llama al evento use para usar el Programa creado, dibujar el objeto objDib y Activar el búfer
                    objGCS.use(objDib,alVertices.size(),ventana);
                    glfwPollEvents(); //capturar cualquier interacción del usuario con la ventana
                }
```

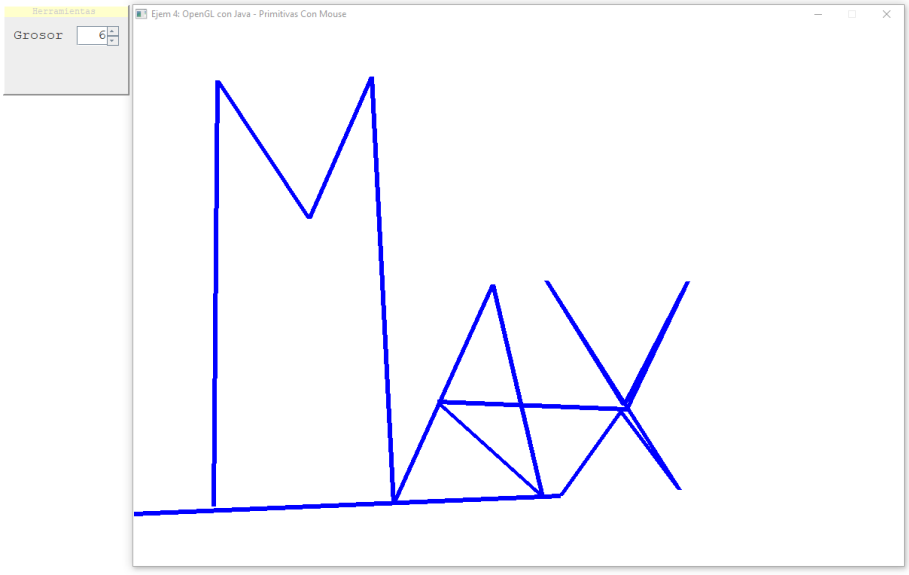
```
92                  glDeleteVertexArrays(objDib);
93                  glDeleteBuffers(objDib);
94                  glfwFreeCallbacks(ventana); //Liberar las llamadas que interactuen con la ventana
95                  glfwDestroyWindow(ventana); //Liberar memoria de la ventana creada
96              }
97          } finally {
98              glfwTerminate(); //limpia y termina la aplicación
99          }
100     }
101 }
102
```

El objeto *objDib* se manda con datos nulos de los arreglos de lista para inicializar la generación de puntos; el evento que permite interactuar con el cursor (mouse) es: *glfwSetCursorPosCallback*; cada vez que se mueva el mouse se ejecutará este evento, el cual agregará las coord. *posX*, *posY* convertidas con el evento *cPAV*, pero al siguiente refresco de la ventana se borrará; este proceso es necesario pues al hacer el primer clic se ingresarán las coord. de inicio de la línea y por ende la condición de borrado dependerá del objecto: *objCM.getEsPosIni()*; pues al ser "1" indica que se tiene un *punto inicio* y su siguiente punto (*punto final*) será el movimiento del mouse y este movimiento se ira *ingresando en la cola* de los arreglos de listas y a la vez borrándose, haciendo el efecto de línea dinámica, la cual quedará fija hasta hacer otro clic en el mouse. (fuente propia - aporte)

La razón de usar el *objCM* como clase, recae en el evento *glfwSetMouseButtonCallback* pues estos Callback usan la inserción de código asíncrono (tipo AJAX) y llamar a una variable dentro de esta sección esta prohibido por ser una variable *LAMBDA*, es así que se utiliza la abstracción de clases, las cuales si están permitidas, por lo tanto; al hacer clic y soltar el botón (*GLFW_RELEASE*) , se ejecutará la asignación del valor a 1 (*objCM.getEsPosIni(1)*), indicando que se agregará el punto inicio de una línea.

El objeto *objDib* nuevamente toma protagonismo, debido a que este se encargará de obtener el conglomerado de puntos de forma dinámica, generada por el accionar del mouse, por ello es que *entra en el ciclo* preguntando si el mouse generó nuevos puntos por el movimiento de este, esto evaluado mediante el evento *glfwPollEvents()*. Por lo demás, ya se explicó el código restante.

- Ejemplo de la aplicación:



- **NOTA COMPLEMENTARIA:** se investigó varias maneras de poder integrar las ventanas en un contenedor Principal, pero lamentablemente la programación GLFW con Java siempre genera una ventana de dibujo Independiente, es así que se pudo idear la forma de manejar múltiples ventanas que gestionaran sus datos gracias a la abstracción de clases.

  Con programación C++, esto sí se puede realizar; Pero se investigó aún más para Java, encontrando tutoriales, que utilizan otras clases de programación como son TWL (aporte de programador) o con GL4Java o con Canvas.

- Material de apoyo que da ideas del uso de GLFW y callBacks con Java
  - https://www.youtube.com/watch?v=EE5cS8EMT78
  - https://www.youtube.com/watch?time_continue=49&v=3WQmbAkwyZk