- Proyecto que dibuje una primitiva geométrica (Ejemplo 02):
    - Se creará el proyecto jOpenGLEjem2
    - Se agregarán las librerías GLFW y OpenGL
    - Programando la aplicación
        - Tal igual como el ejemplo anterior se trabajará con una clase ***"vcd"*** que permitirá gestionar los mensajes de alerta a la hora de crear una ventana; de la misma forma se usará el evento inicializador: ***ini_GLFW*** solo que en este caso necesitamos agregar la línea *GL.createCapabilities()* (encargada de vincular la interacción del *contexto GLFW* con *OpenGL*) después de asignar el contexto al hilo de proceso actual:

```
    glfwSetWindowPos(ventana,(vidmode.width() - ancho) / 2,(vidmode.height() - alto) / 2); //Centr

    glfwMakeContextCurrent(ventana); //Colocar la ventana como hilo de proceso actual
    glfwSwapInterval(1); // Activar la sincronización vertical en el proceso actual(1), para evita

    GL.createCapabilities(); //Se procede a vincular la interación del contexto GLFW con openGL

    objVCD.setVentana(ventana); //Asignar la ventana creada al objeto objVCD para su retorno (en e
    return objVCD;
}
```

        - Se creará el *Programa Sombreador* (*ShaderProgram*) visto en VS. En la cual se debe gestionar:
            - La compilación en tiempo de ejecución
            - Generación y verificación del Vértice y Fragmento
            - Vinculación de sombreadores que fenece en el programa sombreador
            - y Si se rellena el objeto o se crea una malla alámbrica
        - De la misma forma, visto en VS. Debe manejarse una estructura (C++) de nombre ***"pscd"***, pero mediante una *clase Java*, que gestione los mensajes de alerta del *ShaderProgram*:

```
 * and open the template in the editor.
 */
package jopenglejem2;

/**
 *
 * @author LAPTOP-MAX
 */
public class pscd {
    private String msj;
    private String infoLog;
    private  int progSomb;

    public pscd(){
        msj = "";
        infoLog = "";
        progSomb = 0;
    }
    public String getMsj() {
        return msj;
    }

    public void setMsj(String m) {
        msj = m;
    }
```

```
    public String getInfoLog() {
        return infoLog;
    }

    public void setInfoLog(String il) {
        infoLog = il;
    }

    public  int  getProgSomb() {
        return progSomb;
    }
    public void setProgSomb( int ps) {
        progSomb = ps;
    }
}
```

- Se construye el evento *genProgSombreador* tal igual se hizo en VS. Pero en este caso debe tomarse en cuenta las siguientes modificaciones:

```
private static pscd genProgSombreador(String color, boolean relleno){
    pscd objPSCD = new pscd();

    //Se definen las variables de compilación para un vértice y un fragmento
    String vertexShaderSource =
    "#version 330 core\n"+
    "layout(location = 0) in vec3 aPos;\n"+
    "void main()\n"+
    "{\n"+
        "gl_Position = vec4(aPos.x,aPos.y,aPos.z,1.0);\n" +
    "}\0";

    String fragmentShaderSource =
    "#version 330 core\n"+
    "out vec4 FragColor;\n"+
    "void main()\n"+
    "{\n"+
        "FragColor = vec4("+color+");\n" +
    "}\n\0";

    //Se construye y compila el Programa Sombreador
    //------------------------------------------
    //Vértice sombreador
    int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, vertexShaderSource);
    glCompileShader(vertexShader);

    //Verificar si el vertexShader tuvo éxito al compilarse
    int success = glGetShaderi(vertexShader, GL_COMPILE_STATUS);
    if (success == GL_FALSE){
        objPSCD.setMsj("ERROR:Vértice sombreador::Falló la compilación\n");
        int infoLogLength = glGetShaderi(vertexShader, GL_INFO_LOG_LENGTH);
        objPSCD.setInfoLog(glGetShaderInfoLog(vertexShader, infoLogLength));
        return objPSCD;
    }

    //Fragmento sombreador
    int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, fragmentShaderSource);
    glCompileShader(fragmentShader);

    //Verificar si el fragmentShader tuvo éxito al compilarse
    success = glGetShaderi(fragmentShader, GL_COMPILE_STATUS);
    if (success == GL_FALSE){
        objPSCD.setMsj("ERROR:Fragmento sombreador::Falló la compilación\n");
        int infoLogLength = glGetShaderi(fragmentShader, GL_INFO_LOG_LENGTH);
        objPSCD.setInfoLog(glGetShaderInfoLog(fragmentShader, infoLogLength));
        return objPSCD;
    }

    //Vincular Sombreadores
    int shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
```

```
        //Verificar si el shaderProgram tuvo éxito al compilarse
        success = glGetProgrami(shaderProgram, GL_LINK_STATUS);
        if (success == GL_FALSE){
            objPSCD.setMsj("ERROR:Vinculación de sombreadores::Falló la compilación\n");
            int infoLogLength = glGetProgrami(shaderProgram, GL_INFO_LOG_LENGTH);
            objPSCD.setInfoLog(glGetProgramInfoLog(shaderProgram, infoLogLength));
            glDeleteShader(vertexShader);
            glDeleteShader(fragmentShader);
            return objPSCD;
        }
        glDeleteShader(vertexShader); //liberar memoria del contenido del vértice
        glDeleteShader(fragmentShader); //liberar memoria del contenido del Fragmento

        if(!relleno){glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);}

        objPSCD.setMsj("");
        objPSCD.setInfoLog("");
        objPSCD.setProgSomb(shaderProgram);
        return objPSCD;
    }
```

Se instancia el *objPSCD* para la gestión de alertas a la hora de crear los shader.

1. Compilaciones en tiempo de ejecución: como se explicó en los ejercicios de VS. Se crearán 2 cadenas (*vertexShaderSource*; *fragmentShaderSource*) que lleven las instrucciones para compilar la gestión de coord. 3D y el color de dibujo para la primitiva a ingresar; en esta parte de código es muy práctico Java debido a que no se complica con el manejo de punteros a cadenas; todo esto se gestiona con la clase String de manera sencilla.

2. Generación y verificación del Vértice y Fragmento: Se dependerá de las variables *vertexshader* y *fragmentshader* los cuales alertaran si hubo éxito en su compilación; aquí el código cambia en algo; esto debido a que, C++ maneja eventos que tienen agregados recursivos, en cambio Java, como toda función; regresa un resultado, esto se aprecia cuando se usa la variable *infoLogLength* y se devuelve su resultado en el objeto de clase PSCD: *setInfoLog*.

3. Vinculando los Shader: En esta parte el código es igual que en C++, con la diferencia del evento recursivo, que en este caso tiene forma de función (*glGetProgrami*); de todas maneras se hace un control de alertas y posteriormente se libera memoria de los Shader utilizados.

4. Indicar relleno o malla alámbrica: mientras que en C++ los eventos son tratados con GLAD, en Java; el evento *glPolygonMode* depende de las librerías OpenGL (GL<…>). Finalmente se retornan los resultados mediante el objeto *objPSCD.*

- El evento *genObjLinADib* tiene similitud con C++, solo que, en Java; la memoria de los objetos no se manipula en forma directa; es por ello que, el código cambia:

```
private static int genObjLinADib(float xini,float yini,float xfin,float yfin){
    float vertices[] = {
        xini, yini, 0.0f,
        xfin, yfin, 0.0f
    };

    int VAO = 0, VBO = 0;
    VAO = glGenVertexArrays();
    VBO = glGenBuffers();
    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);

    FloatBuffer verticesBufferObject = BufferUtils.createFloatBuffer(vertices.length); // crear contenedor
    verticesBufferObject.put(vertices); // poner los datos del arreglo en el contenedor
    verticesBufferObject.flip(); // Limpiar el índice del contenedor (método similar a trim() de cadenas)

    glBufferData(GL_ARRAY_BUFFER, verticesBufferObject, GL_STATIC_DRAW);
    glVertexAttribPointer(0,3,GL_FLOAT,false,0,0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
    return VAO;
}
```

para *glBufferData*: En C++ se establece la memoria del vector *vertices[]*, pero para Java se usa:

FloatBuffer verticesBufferObject =
BufferUtils.createFloatBuffer(vertices.length);

El cual debe crear un *búfer contenedor de tipo Float (createFloatBuffer)*, pero como se debe pasar datos de un array (*vertices[]*), se usa la función *BufferUtils*.

verticesBufferObject.put(vertices);
verticesBufferObject.flip();

Creado el contenedor, se debe colocar los datos del array mediante el método *put(<array>)*. Acto seguido se debe gestionar la limpieza de sus índices (algo parecido al método *trim() de cadenas)* mediante el método ***flip().***

Con la gestión de código anterior, ahora se puede ingresar información al evento

glBufferData(GL_ARRAY_BUFFER, verticesBufferObject, GL_STATIC_DRAW);

a diferencia de C++ que lleva 4 parámetros; en Java se reduce a 3, debido a que *verticesBufferObject* ya envía el tamaño del array y los datos de este.

glVertexAttribPointer Ya fue definido a grandes rasgos en C++, pero debe tenerse en cuenta que, al comprender los 2 últimos parámetros desde Java, estos valores van en 0, debido a que el 5to parámetro (Stride) = 0 indica que los datos del vértice están estrechamente empaquetados, es decir; se asume que cada 3 posiciones se tiene un vértice y sucesivamente y el 6to parámetro (offset) es el valor inicial del arreglo (algo complejo de entender).

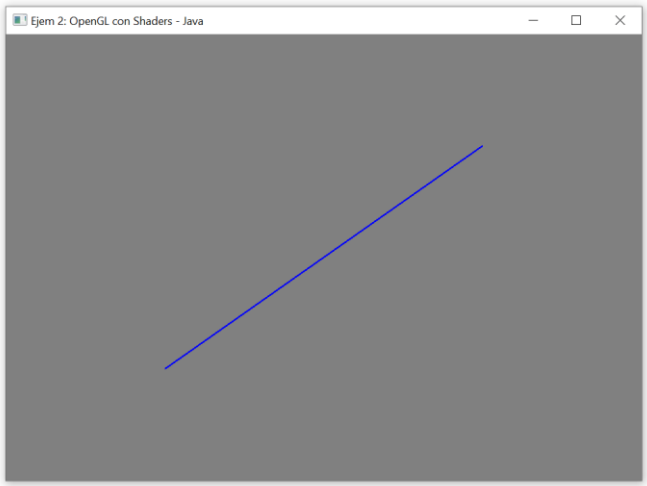
Las demás líneas de código restantes, ya se sobrentienden pues tienen el mismo significado que en C++ (explicado).

- Programando entonces, el "**Main**":

```
169     public static void main(String[] args) {
170         long ventana = NULL;
            vcd venConDat = new vcd();
            pscd progSombConDat = new pscd();
173         int progSomb = 0;
            int objDib = 0;
175         try {
176             //Se inicializa GLFW, asimismo si todo es correcto, se crea la ventana
177             venConDat = ini_GLFW(ventana,1024, 720, "Ejem 2: OpenGL con Shaders - Java");
178             if (!"".equals(venConDat.getMsj())) {
179                 System.out.println(venConDat.getMsj()); //imprime el error cometido
180                 glfwTerminate(); //limpia y termina la aplicación
181             }
182             else {
183                 ventana = venConDat.getVentana();
184             }
185
186             //Se construye el Programa Sombreador (ShaderProgram) asignando el color de línea
187             progSombConDat = genProgSombreador("0.0f, 0.0f, 1.0f, 1.0f", false);
188             if (!"".equals(progSombConDat.getMsj())) {
189                 System.out.println(progSombConDat.getMsj()); //imprime el msg del error cometido
190                 System.out.println(progSombConDat.getInfoLog()); //imprime el log del error cometido
191                 glfwTerminate(); //limpia y termina la aplicación
192             }
193             else {
194                 progSomb = progSombConDat.getProgSomb();
195             }
196
197             objDib = genObjLinADib(-0.5f, -0.5f, 0.5f, 0.5f);
198
199             while (!glfwWindowShouldClose(ventana)) {
200                 glClearColor(0.5f, 0.5f, 0.5f, 0.0f); //Configurar color de fondo
201                 glClear(GL_COLOR_BUFFER_BIT); // limpiar el FrameBuffer
202                 glUseProgram(progSomb);
203                 glBindVertexArray(objDib);
204                 glDrawArrays(GL_LINE_STRIP, 0, 2);
205                 glLineWidth(3.0f);
206
207                 glfwSwapBuffers(ventana); //activar el doble búfer
208                 glfwPollEvents(); //capturar cualquier interacción del usuario con la ventana
209             }
210             glDeleteVertexArrays(objDib);
211             glDeleteBuffers(objDib);
212
213             glfwFreeCallbacks(ventana); //Liberar las llamadas que interactuen con la ventana
214             glfwDestroyWindow(ventana); //Liberar memoria de la ventana creada
215         } finally {
216             glfwTerminate(); //limpia y termina la aplicación
217         }
218     }
219 }
220
```

Él código anterior es muy similar al de C++ (ejercicios anteriores); por tanto, se asume que ya fué explicada.

- El resultado de ejecutar dicha aplicación sería:



- Material de apoyo que da ideas del uso de GLFW y OpenGL con Java
  - https://community.khronos.org/t/glviewport-java-lwjgl/70928