```
// hello.c++: A "Hello, OpenGL" program that draws a triangle
// This is a standalone version that does not make use of our MVC framework.

#include <iostream>
#include <stdlib.h>
#include "GLFW/glfw3.h"
#include "ShaderIF.h"

GLFWwindow* theWindow;
ShaderIF* sIF = NULL;
GLuint vao[1];
GLuint vbo[1];

void createWindowAndRC()
{
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
    theWindow = glfwCreateWindow(400, 400, "Hello, OpenGL", NULL, NULL);
    if (theWindow == NULL)
    {
        std::cerr << "Could not create a 4.1 OpenGL Rendering context.\n";
        exit(1);
    }
    glfwMakeContextCurrent(theWindow);
}

void handleDisplay(GLFWwindow* window)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glUseProgram(sIF->getShaderPgmID());

    glBindVertexArray(vao[0]); // Make our VAO-VBO "active"
    // Starting at location 0 in the VBO, use 3 vertices to draw a triangle:
    glDrawArrays(GL_TRIANGLES, 0, 3);

    glfwSwapBuffers(window);
}
```

Controller instance variable

ModelView instance variables

Actual creation of RC

ModelView::render()

The Controller will loop over all ModelView instances, calling the render method of each.

Controller instance methods

ModelView instance method

```
void initModelGeometry()
{
    float triangleVertices[3][2] =
    {
        { -0.75, -0.75 }, { 0.75, -0.75 }, { 0.0, 0.75 }
    };

    glGenVertexArrays(1, vao); // get a new, previously unused VAO name
    glBindVertexArray(vao[0]); // (initialize it and) make it active

    glGenBuffers(1, vbo); // get a new, previously unused VBO name
    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]); // (initialize it and) make it active

    // Allocate space for AND copy the CPU data in "triangleVertices" to the VBO
        on the GPU:
    int numBytesInBuffer = 3 * 2 * sizeof(float); // 3 points; 2 coords each;
        float
    glBufferData(GL_ARRAY_BUFFER, numBytesInBuffer, triangleVertices,
        GL_STATIC_DRAW);

    int coordinateLocation = 0; // See "layout" in hello.vsh
    GLsizei stride = 0; // "tightly packed"
    void* offset = 0;   // start at beginning of currently bound VBO.
    glVertexAttribPointer(coordinateLocation, 2, GL_FLOAT, GL_FALSE, stride,
        offset);
    glEnableVertexAttribArray(coordinateLocation);
}
```

```
void run()
{
    while (!glfwWindowShouldClose(theWindow))
    {
        glfwWaitEvents();
        handleDisplay(theWindow);
    }
    glfwDestroyWindow(theWindow);
}
```

Controller instance Method

```
int main(int argc, char* argv[])
{
    glfwInit();             // 0. Initialize GLFW window system
    createWindowAndRC(); // 1. Create window and Rendering Context (RC)
    sIF = new ShaderIF("hello.vsh", "hello.fsh");
                            // 2. Create the shader interface
    initModelGeometry(); // 3. Create and send geometry of model to GPU
    run();                  // 4. Hand off to window manager to monitor events

    // Program is over. Clean up.
    glDeleteBuffers(1, vbo);
    glDeleteVertexArrays(1, vao);
    delete sIF;

    return 0;
}
```

done in Controller

done in ModelView destructor

② done in ModelView Constructor

ⓑ The instance is registered with the Controller using the "addModel" method.