**Введение.**

Разбор главы о LWJGL 3 является логическим продолжение главы о LWJGL 2. Настоятельно рекомендую хотя бы просто прочитать (без практики) главу по LWJGL 2 т.к. освещены многие важные моменты, которые переносятся на LWJGL 3 и в этой главе повторно не объясняются.

**Установка.**

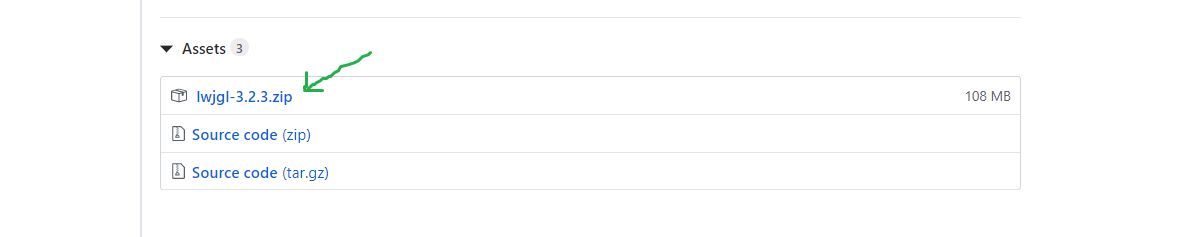
Информация, представленная здесь основана на документации lwjgl 3 по разделу «Install»:

<https://github.com/LWJGL/lwjgl3-wiki/wiki/1.2.-Install>

**Скачивание библиотеки (всех ее модулей).**

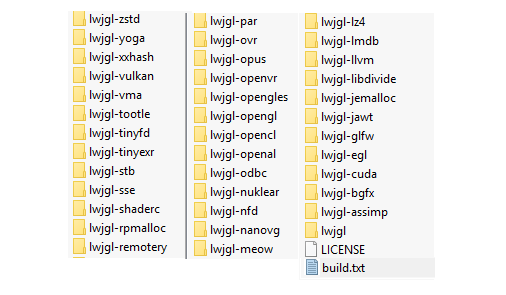
Скачайте по ссылке <https://github.com/LWJGL/lwjgl3/releases> архив с последней версией библиотеки.

Я скачиваю версию 3.2.3:



В скачанном архиве содержатся все модули библиотеки. Распакуйте архив в какую-нибудь папку.

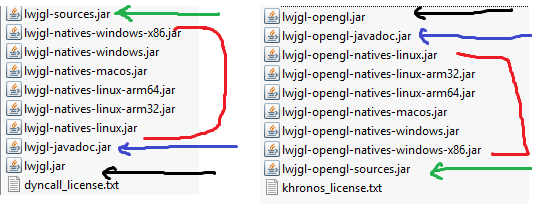
Библиотека LWJGL 3 состоит из модулей. Каждый модуль имеет свои jar архивы, отвечающие за определенный функционал библиотеки:



Основной модуль называется просто lwjgl (последняя папка в 3ей колонке на рисунке).

В каждом модуле (папке) содержатся jar архивы, именованные по определенному смыслу.

На примере модулей lwjgl и lwjgl-opengl:



* Зеленый – Java исходники.
* Красный – нативные библиотеки (для работы на определенных ОС).
* Синий – документация.
* Черный – скомпилированные Java классы.

Вы можете скачивать и использовать только те модули, которые требует ваш проект.

**Есть еще один способ скачать библиотеку (выборочно по модулям):**

Официальный сайт lwjgl имеет помощника, который предложит вам скачать необходимые модули по выбранным вами настройкам: <https://www.lwjgl.org/customize>

Для опытных разработчиков есть возможность использовать системы сборок Maven или Gradle.

Для избегания различных багов рекомендуется использовать версию Release.

**Замечание.**

Так как мы не пользуемся функционалом всех модулей и .jar архивов из библиотеки, то

рекомендовано добавлять только те .jar архивы, которые требуются вашему приложению (чтобы не использовать лишнюю память или избегать некоторые конфликты с совместимостями, если ваше приложение будет сложным и замудренным).

В главе по lwjgl мы не заморачивались и просто использовали все .jar архивы всей библиотеки. В lwjgl 3 мы так не поступим т.к. их объем превысит сотню мегабайт. Добавим только те модули, которые понадобятся в этой методичке.

Для этого создайте папку с названием lib и переместите в нее все .jar архивы из модулей:

* lwjgl
* lwjgl-opengl
* lwjgl-glfw

**JOML – Java OpenGL Math Library.**

Также нам понадобится математическая java библиотека JOML.

JOML предоставляет возможности (классы, методы) для использования линейной алгебры (работа с матрицами, векторами и т.д.).

**Репозиторий библиотеки:**

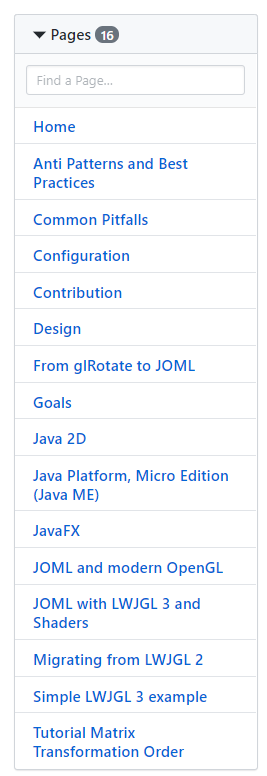
<https://github.com/JOML-CI/JOML>

Рекомендую прочитать описание (файл README.md)

**Документация по JOML:**

<https://github.com/JOML-CI/JOML/wiki>

На странице вы найдете меню с разделами (рекомендую самостоятельно их изучить):



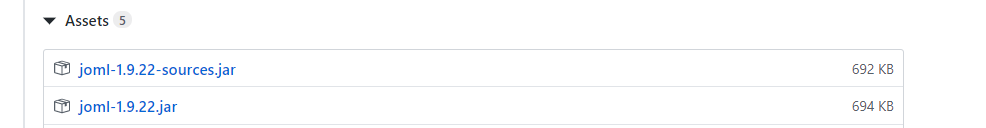
Как минимум рекомендую освоить разделы:

****

**Установка.**

По ссылке <https://github.com/JOML-CI/JOML/releases>

скачайте эти .jar архивы и переместите их в папку lib:

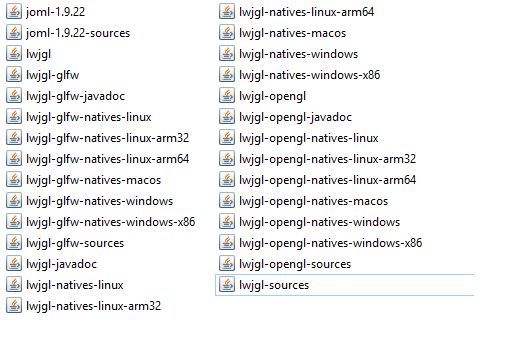


**Создание первого LWJGL 3 проекта.**

Далее, я буду использовать IntelliJ IDEA. Но если вы пользуетесь другими средами разработки, то вы можете найти гайды по ним на сайте документации:

<https://github.com/LWJGL/lwjgl3-wiki/wiki/1.2.-Install>

**Папка lib должна содержать эти .jar архивы:**

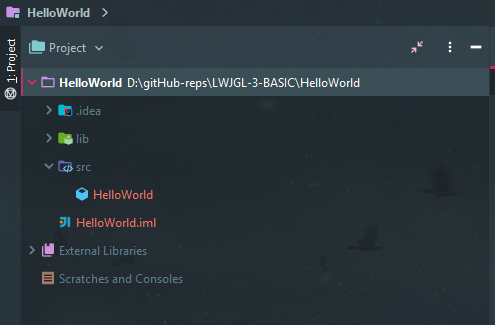


Сохраните эту папку где-нибудь на компьютере. Ее мы будем добавлять в каждый новый проект.

В IntelliJ IDEA создайте новый проект под названием HelloWorld.

Скопируйте папку lib в папку с проектом (не в папку src! А на директорию выше!).

В папке src проекта создайте java класс под названием HelloWorld.



С официального сайта LWJGL 3, с раздела «GET STARTED»:

<https://www.lwjgl.org/guide>

скопируйте код-пример и вставьте его в созданный файл HelloWorld.

Сущности кода должны подсвечиваться красным, сигнализируя об ошибке. Это потому что мы не связали библиотеку lib с проектом.

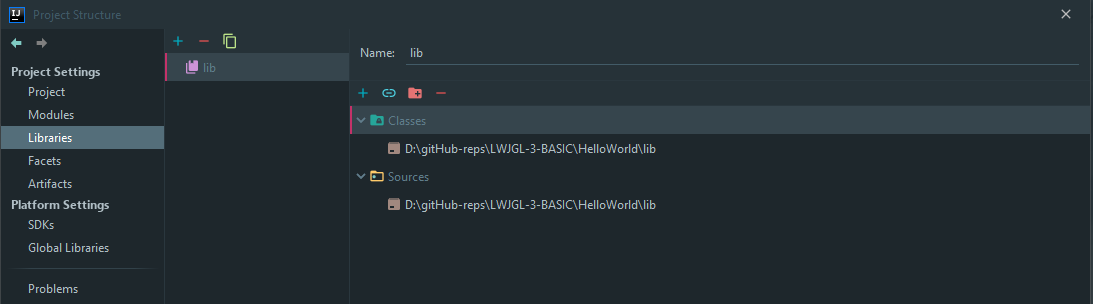
Теперь подключим нашу библиотеку к проекту.

Идем в **File** **->** **Project Structure.** В разделе **Libraries** кликаем на плюсик и выбираем Java. Откроется окно, в котором указываем путь к папке lib, которую вы скопировали в папку с проектом.

У меня путь такой:

D:\gitHub-reps\LWJGL-3-BASIC\HelloWorld\lib

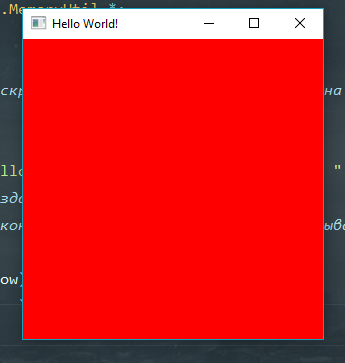
Жмем ok, ok.



Жмем Apply, ok. Библиотеку подключили.

Теперь попробуйте запустить программу.

Если программа запустится, то у вас откроется окно (дисплей) с красным фоном. Его можно закрыть, нажав клавишу «ESCAPE».

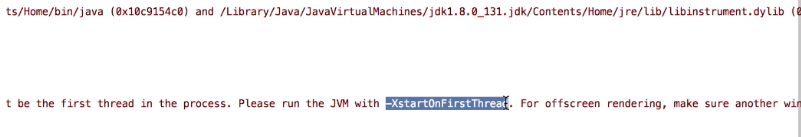


**Замечание.**

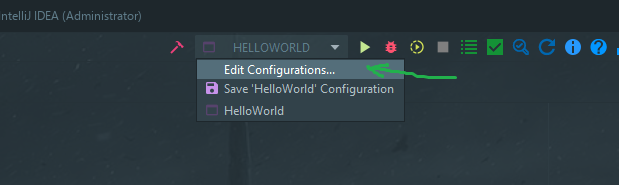
У меня программа запустилась, но у вас программа может не запуститься с такой ошибкой:



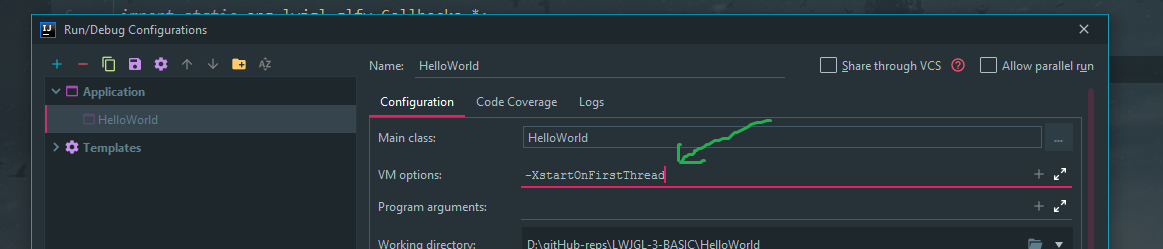
Для ее устранения скопируйте эту строчку «-XstartOnFirstThread»:



Идем сюда:



В открывшемся окне введите скопированную строку в поле VM options:

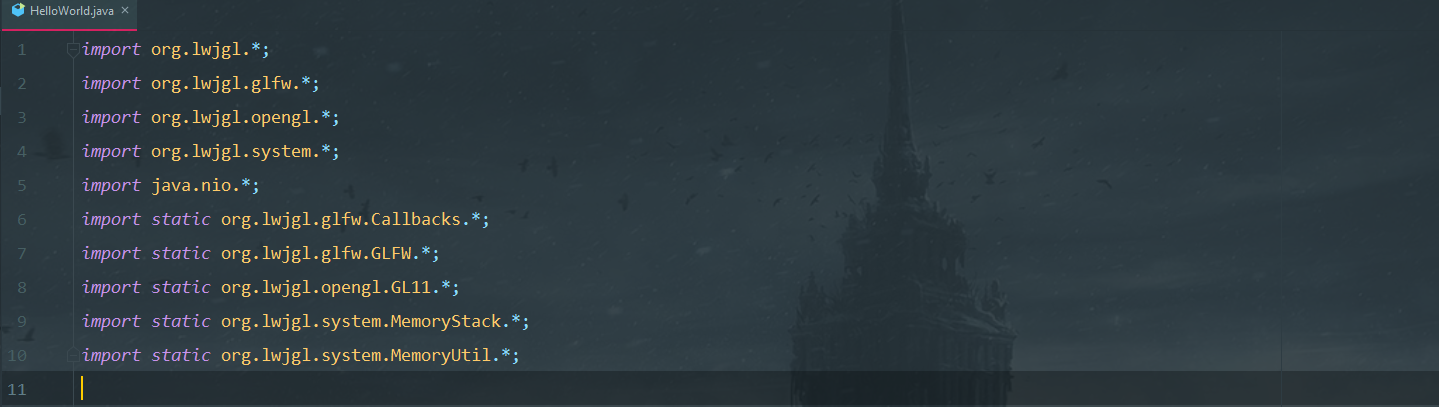


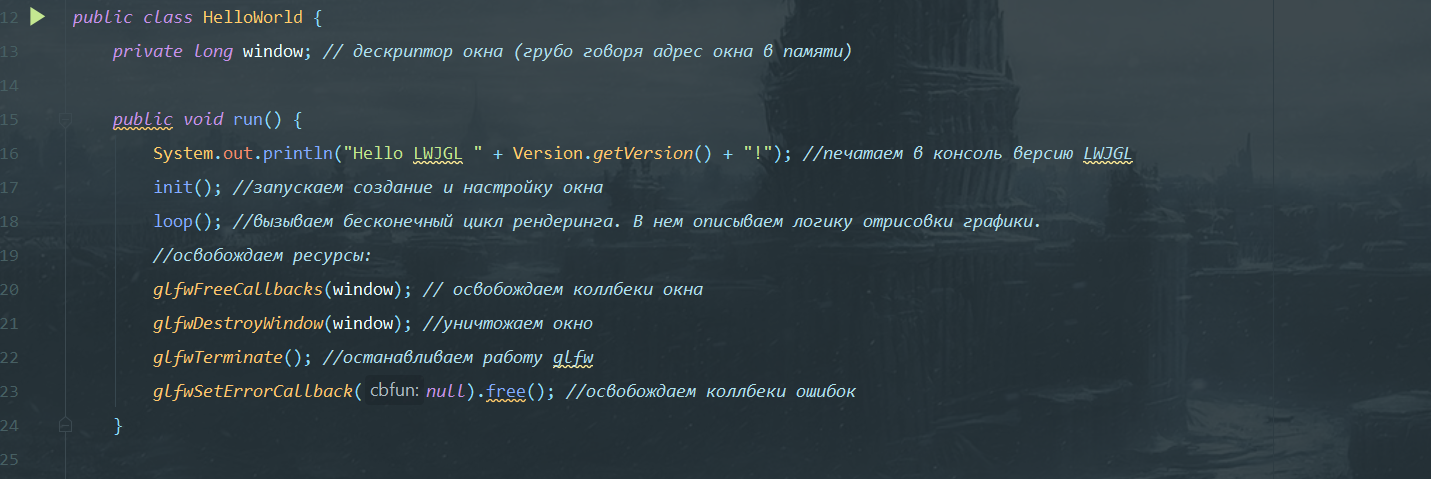
Жмем apply, ok.

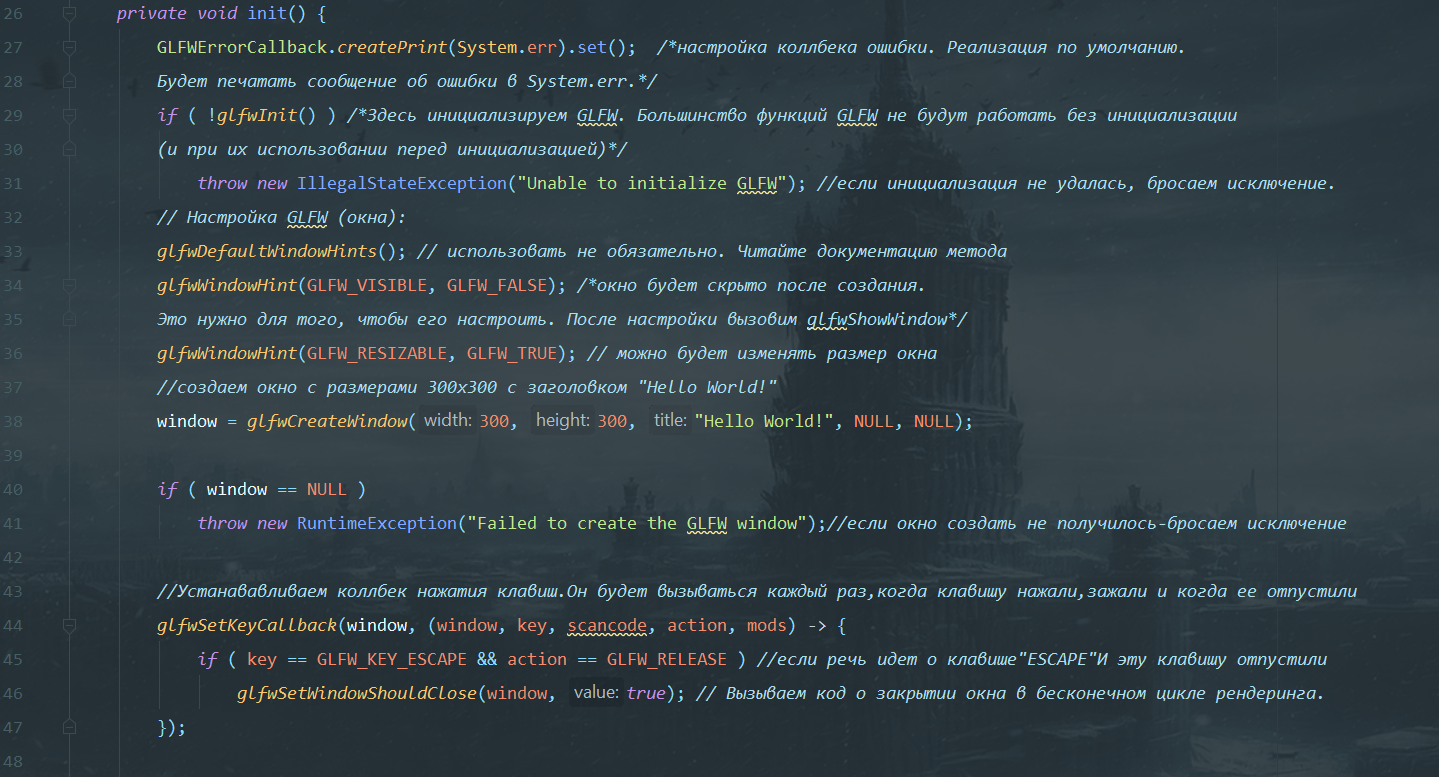
Теперь программа должна запуститься.

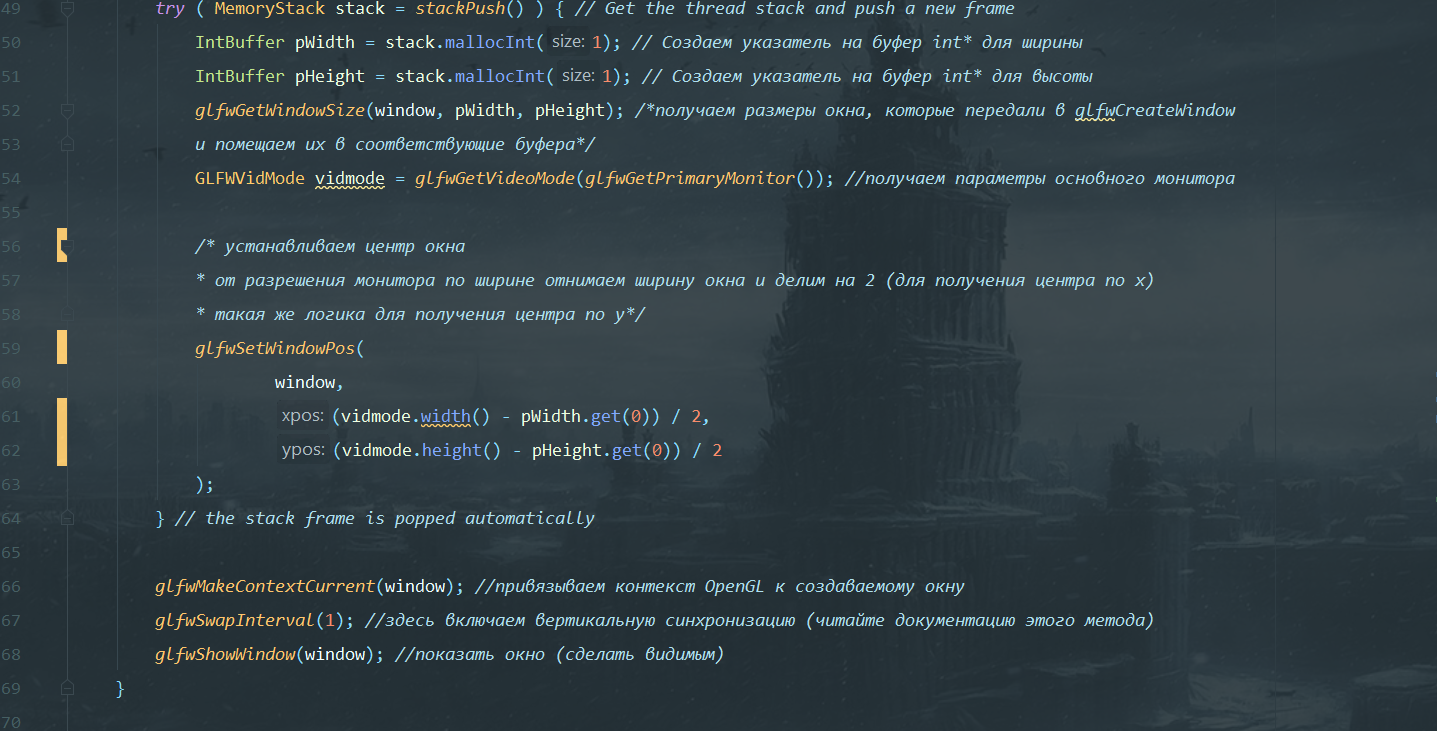
**Теперь разберемся что за код-пример мы использовали.**

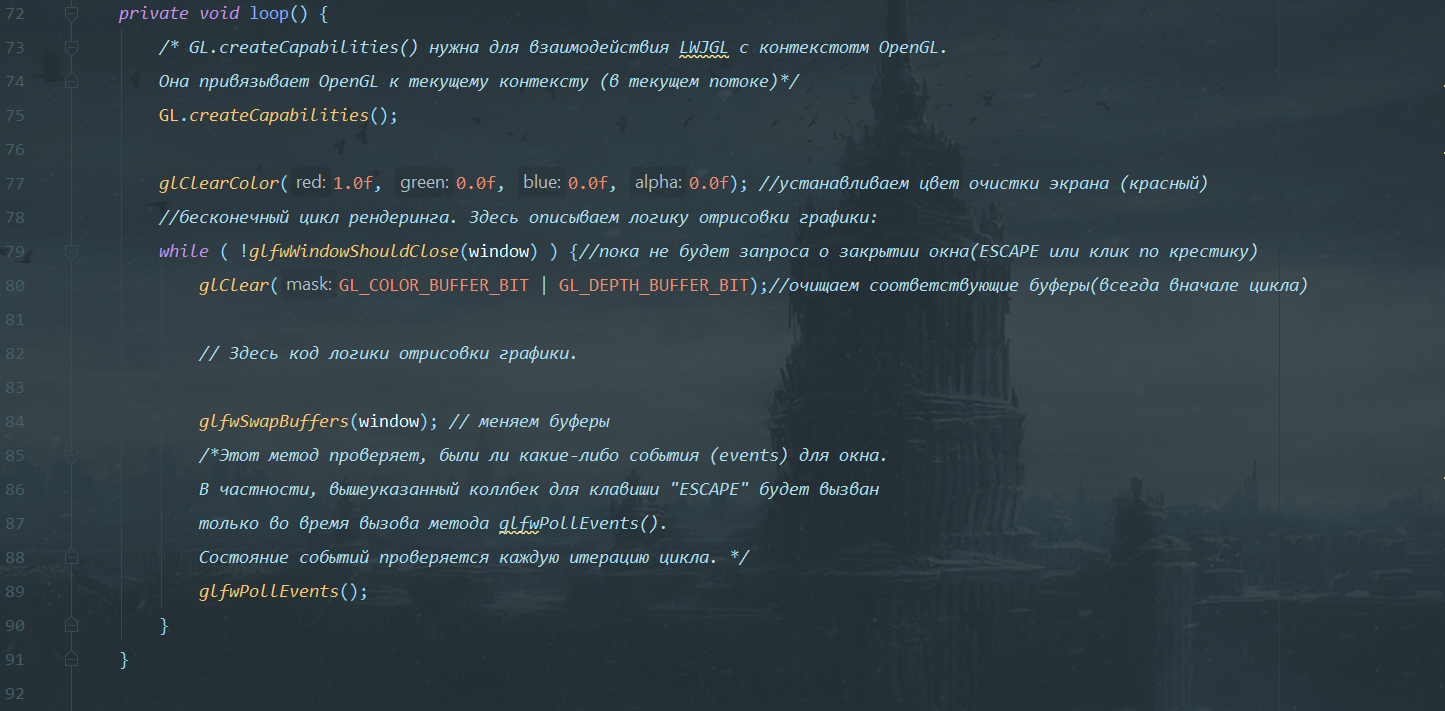
Каждое действие перевел на русский и местами добавил дополнительные комментарии:

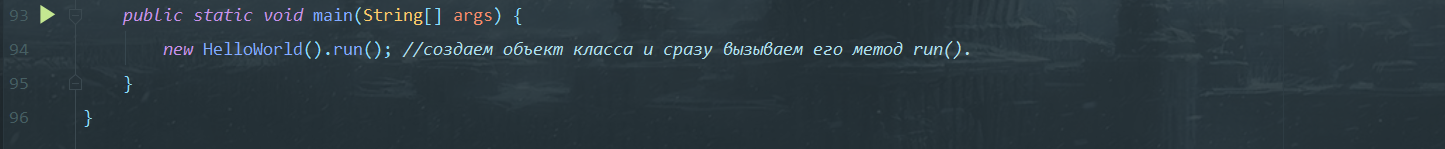












Выглядит страшно только из-за комментариев.

Этот код можно использовать как заготовку для ваших следующих приложений.

Перейдем к дополнительному разбору кода.

В коде-примере используемые библиотеки импортируются статически. Поэтому перед вызовом методов не приписывается «GLFW.», «GL11.», и тд (т.е. обращение к классам).

Про этот момент также отмечалось в разборе LWJGL 2.

В следующих программах-прмирах я буду приписывать обращение к классам. Не запутайтесь.

**Контекст.**

Под контекстом считайте воображаемое окно, в которое отрисовывает OpenGL.

LWJGL связывает свое окно с контекстом (с воображаемым окном OpenGL’а) с помощью метода:

*glfwMakeContextCurrent*(window); *//привязываем контекст OpenGL к создаваемому окну*

Более серьезные определения и попытки объяснить, что такое контекст можете почитать в этой беседе:

<https://gamedev.ru/code/forum/?id=168946>

**GLFW.**

**GLFW** – это библиотека, которая пришла на замену классам **Display, Keyboard, Mouse** из LWJGL 2.

GLFW - это многоплатформенная библиотека с открытым исходным кодом для создания окон с контекстами OpenGL и получения событий (events).

Она легко интегрируется в приложения.

GLFW имеет простой, но мощный API с подробной документацией:

<https://www.glfw.org/docs/latest/>

**Вывод ошибок GLFW в консоль.**

Вывод ошибок GLFW в консоль осуществляется вызовом метода

GLFWErrorCallback.*createPrint*(System.err).set();

вначале метода **init()**.

В конце работы программы - в конце метода **run()** вызывается метод

*glfwSetErrorCallback*(*null*).free(); *//освобождаем коллбеки ошибок*

который очищает буфер ошибок (освобождение ресурсов).

За ошибку инициализации GLFW ресурсов (создания окна) под капотом отвечает конструкция

*if* ( !*glfwInit*() ) */\*Здесь инициализируем GLFW. Большинство функций GLFW не будут работать без инициализации  
(и при их использовании перед инициализацией)\*/  
 throw new* IllegalStateException("Unable to initialize GLFW"); *//если инициализация не удалась, бросаем исключение.*

А также конструкция

*if* ( window == NULL )  
 *throw new* RuntimeException("Failed to create the GLFW window");*//если окно создать не получилось-бросаем исключение*

которая бросает исключение, если в переменную **window** (грубо говоря указатель на окно в памяти. Иногда вместо «window» переменную называют «id») не было присвоено «адекватное» значение вызовом метода:

window = *glfwCreateWindow*(300, 300, "Hello World!", NULL, NULL);

Если создание окна не удалось, то метод **glfwCreateWindow()** присвоит переменной **window** значение 0.

Сейчас расскажу о приколе, который произошел со мной.

После того, как я посмотрел на выражение

*if* ( window == NULL )

я долго не мог понять, на каком основании window сравнивается с NULL. Ведь тип переменной **window** – **double** (примитивный тип), а в JAVA значение NULL никак не может присваиваться примитивным типам данным.

Я даже всерьез посчитал, что в примере допущена ошибка и **window** должна быть типа **Double** (ссылочного типа. Таким типам можно присваивать null).

Но после консультации с парнем на потоке в университете он заметил, что этот NULL – это не тот null, про который идет речь. Ведь в JAVA null пишется маленькими буквами т.е. так “null”, а в данном случае NULL – это константа, которая имеет значение 0L.

Получается, что в условии **window** сравнивается с 0L.

Такой глюк у меня произошел после годичного отдыха от джавы и длительного изучения языков программирования java script и golang.

В golang как раз NULL пишется большими буквами. Отсюда и brain lag.

Не попадайтесь на такие приколы.

**Настройки параметров окна.**

После создания окна можно задавать ему параметры.

* Следующий метод задает параметры по умолчанию для окна:

*glfwDefaultWindowHints*(); *// использовать не обязательно. Читайте документацию метода*

* Следующая строка разрешает изменять размер окна (потянув курсором).

*glfwWindowHint*(GLFW\_RESIZABLE, GLFW\_TRUE); *// можно будет изменять размер окна*

Эту конструкцию выгоднее будет переписать следующим образом.

Добавьте поле в класс:

*private boolean* resizable = *true*;

А сам вызов метода **glfwWindowHint()** перепишите так:

*glfwWindowHint*(GLFW\_RESIZABLE, resizable ? GLFW\_TRUE : GLFW\_FALSE);

Таким образом будет приятнее настраивать параметр окна **GLFW\_RESIZABLE** с помощью присваивания **true** или **false** в переменную **resizable**.

* Конструкцию

window = *glfwCreateWindow*(300, 300, "Hello World!", NULL, NULL);

Также выгодно переписать в предыдущем стиле.

Добавьте следующие поля в класс:

*private* String title = "MyTitle";  
*private int* m\_width = 300;   
*private int* m\_height = 300;

А сам вызов метода **glfwCreateWindow()** перепишите так:

window = *glfwCreateWindow*(m\_width, m\_height, title, NULL, NULL);

**Официальная документация по LWJGL 3:**

<https://javadoc.lwjgl.org/>

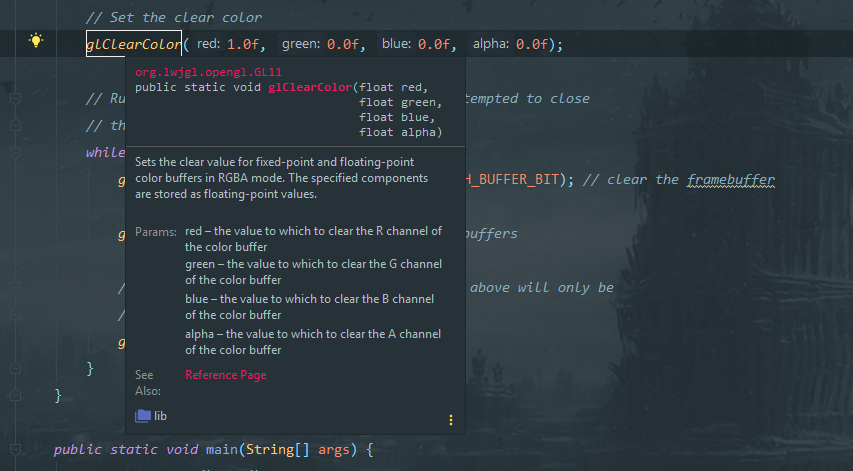
**Официальное wiki (гайды на английском):**

<https://github.com/LWJGL/lwjgl3-wiki/wiki>

**Вывод документации в IntelliJ IDEA.**

Хочу отметить важную новинку. В LWJGL 3 разработчики добавили возможность смотреть документацию по методам, классам, переменным прямо в IDE. Это возможно благодаря написанным Javadoc файлам, которые идут в комплекте с библиотекой.

В IntelliJ IDEA поставьте курсор на интересующий вас метод и нажмите сочетание клавиш «ctrl+q». Откроется окно с описанием метода:



А если нажать сочетание клавиш «ctrl+shift+i», то можно посмотреть реализацию выбранной сущности.

Это может быть полезно, чтобы подглядеть, какие параметры можно передать в метод и в каком порядке.

