**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

СОГЛАСОВАНО  
Профессор департамента программной инженерии, кандидат технических наук  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.М. Гринкруг  
«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

УТВЕРЖДАЮ  
Академический руководитель образовательной программы «Программная инженерия»  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  
«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

**ПРОГРАММНЫЙ КОМПОНЕНТ OPENGLVIEWER БИБЛИОТЕКИ JAVABEANS-КОМПОНЕНТ ДЛЯ 3D-ГРАФИКИ**

**Пояснительная записка**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.01-01 81 01-1-ЛУ**

**Исполнитель**

Студент группы БПИ163

\_\_\_\_\_\_\_\_\_\_\_\_ / Д.Е. Крайнов /

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

**Москва 2019**

**УТВЕРЖДЕН**

**RU.17701729.04.01-01 81 01-1-ЛУ**

**ПРОГРАММНЫЙ КОМПОНЕНТ OPENGLVIEWER БИБЛИОТЕКИ JAVABEANS-КОМПОНЕНТ ДЛЯ 3D-ГРАФИКИ**

**Пояснительная записка**

**RU.17701729.04.01-01 81 01-1**

**Листов 24**

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл |  |

**Москва 2019**

СОДЕРЖАНИЕ

[1. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ 4](#_Toc6425622)

[1.1. Наименование программы 4](#_Toc6425623)

[1.2. Документ, на основании которого ведется разработка 4](#_Toc6425624)

[2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ 5](#_Toc6425625)

[2.1. Функциональное назначение 5](#_Toc6425626)

[2.2. Эксплуатационное назначение 5](#_Toc6425627)

[3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ 6](#_Toc6425628)

[3.1. Постановка задачи на разработку программы 6](#_Toc6425629)

[3.3. Описание алгоритма и функционирования программы 10](#_Toc6425630)

[3.3.1. Описание алгоритма программы 10](#_Toc6425631)

[3.3.2. Описание функционирования программы 11](#_Toc6425632)

[3.3.2.1. Пакет objects и класс GLObject 11](#_Toc6425633)

[3.3.2.2. Пакет viewer и класс OpenGLViewer 12](#_Toc6425634)

[3.3.2.3. Пакет window и класс OpenGLTestFrame 12](#_Toc6425635)

[3.4. Описание и обоснование выбора метода организации входных и выходных данных 12](#_Toc6425636)

[3.4.1. Описание и обоснование выбора метода организации входных данных 12](#_Toc6425637)

[3.4.2. Описание и обоснование выбора метода организации выходных данных 13](#_Toc6425638)

[3.5. Описание и обоснование выбора технических и программных средств 13](#_Toc6425639)

[4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ 14](#_Toc6425640)

[4.1. Ориентировочная экономическая эффективность 14](#_Toc6425641)

[4.2. Предполагаемая потребность 14](#_Toc6425642)

[4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами 14](#_Toc6425643)

[5. ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ 15](#_Toc6425644)

[Приложение 1 16](#_Toc6425645)

[Приложение 2 17](#_Toc6425646)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 24](#_Toc6425647)

# ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

* 1. Наименование программы  
     Наименование программы: «Программный компонент OpenGLViewer библиотеки JavaBeans-компонент для 3D-графики» («The OpenGLViewer Software Component for JavaBeans 3D Graphics Components Library»). Краткое название: “OpenGLViewer”.
  2. Документ, на основании которого ведется разработка  
     Приказ декана факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики» № 2.3-02/1012-0 2 от 10.12.2018.

# НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

* 1. Функциональное назначение  
     Функциональным назначением компонента является предоставление возможности графического рендеринга моделей, представленных в виде графа сцены [1], с помощью технологии OpenGL [2] и графического процессора пользовательского компьютера.
  2. Эксплуатационное назначение  
     Программный компонент предлагается к эксплуатированию программистами в области компьютерной графики и 3D-моделистами в качестве инструмента для тестирования созданных графических моделей.

# ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

* 1. Постановка задачи на разработку программы  
     Компонент должен предоставить интерфейс для взаимодействия с ним другим JavaBeans-компонентам библиотеки, в составе которой он должен работать. В частности, компонент OpenGLViewer должен:
* получать объект 3D-сцены (граф сцены) для отображения;
* поддерживать собственный вид на переданную ему сцену;
* обеспечивать графическое отображение (рендеринг) этого вида на экране, используя возможности библиотеки OpenGL с помощью программного интерфейса Java OpenGL (JOGL) [3];
* предоставлять возможность параллельного наблюдения сцены с помощью нескольких экземпляров компонента;
* предоставлять возможность манипулирования сценой с помощью методов получения и модификации графа сцены и её настроек (геттеры и сеттеры).
  1. **Описание применяемых методов и алгоритмов**
* **Компонентная модель и спецификация JavaBeans**

Компонентно-ориентированное программирование – парадигма программирования, основная идея которой – представление и разработка программ как набора компонентов - независимых модулей, направленных на выполнение одной определенной задачи. Каждый компонент компилируется и подключается к программному проекту отдельно от остальных.

Выделяется несколько основных преимуществ данного подхода [4].

Во-первых, код, написанный с использованием компонентов, проще разрабатывать. В силу ограниченности действий, выполняемых каждым компонентом, становится проще понимать, как добиться той функциональности, какая требуется в финальной программе. По этой же причине легко создавать новые вариации программ с дополнительными функциями путём замены или добавления компонентов.

Во-вторых, изолированность и заточенность компонентов под конкретное действие позволяет проще отслеживать ошибки, возникающие в результате работы.

Наконец, компоненты проще тестировать, так как сразу понятно, что должен и не должен делать компонент.

Спецификация JavaBeans – по сути, реализация компонентной методологии в среде Java [5]. Bean – любой класс, написанный по нескольким правилам:

* в классе присутствует публичный конструктор без параметров;
* все поля – приватные, работа с ними осуществляется через публичные методы доступа (геттеры) и модификации (сеттеры);
* класс имплементирует интерфейс Serializable (должна присутствовать возможность преобразования экземпляра класса в поток байтов для сохранения в файловую систему, базу данных и т.п.);
* геттеры не принимают параметров, возвращают объект определенного типа, названы в стиле get\*(), где \* - название поля;
* сеттеры принимают параметр, ничего не возвращают, названы в стиле set\*(), где \* - название поля.

Подобная стандартизация классов нашла своё применение, в первую очередь, в приложениях, предназначенных для визуального построения программ. Повторное использование компонентов и возможность их конфигурации на лету, а также возможность использования событийной модели (оповещения при изменении свойств класса) – основные преимущества спецификации JavaBeans. В частности, виджеты библиотеки Swing для создания графических интерфейсов в Java являются «бинами».

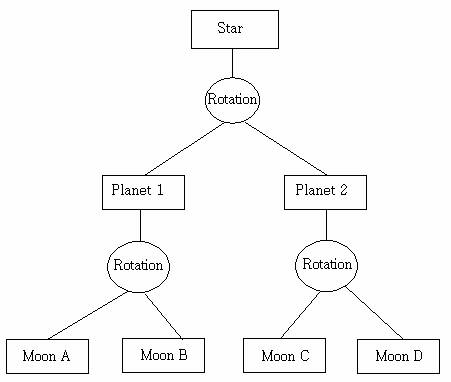
Набор компонентов OpenGLViewer разрабатывается в соответствии с данной спецификацией, что позволит использовать компоненты набора – двух- и трехмерные объекты и сцены, состоящие из данных объектов, а также экран просмотра и вспомогательные компоненты – в визуальных редакторах интерфейсов, фактически – без необходимости написания какого-либо кода.

* **Представление модели в виде графа сцены**

Любая графическая модель складывается из множества графических примитивов. Примитив – простейший геометрический объект, отображаемый на экране. Например, два прямоугольных треугольника могут образовать прямоугольник, множество квадратов в трехмерном пространстве – куб или сферу, и так далее.

В наборе разрабатываемых компонентов, помимо обычных примитивов (линия, треугольник), присутствуют сложные примитивы, упрощающие работу по созданию модели – куб, сфера, цилиндр и т. п. Для их получения не требуется описание множества более мелких примитивов, обычно их составляющих.

В свою очередь, экземпляры компонентов соединяются в одну структуру для удобства моделирования и отображения. Широко распространена практика использования в качестве такой структуры дерева, или «графа сцены».



*Рисунок 1. Пример графа сцены*

На рис. 1 изображен простой граф сцены на примере планетной системы. Есть центральный, коренной объект – звезда (“Star”). Ко всем потомкам – объектам, логически связанным со звездой – применяется трансформация вращения вокруг родительского объекта (“Rotation”). В коде объектов применение вращения может быть запрограммировано как изменение координат планет, а параметры изменения можно получить как композицию информации о трансформации, примененной к данному объекту, и информации о трансформации объекта-родителя, чем будет обеспечиваться логическая связность модели.

Для разработки графов сцены нет какого-либо стандарта, поэтому в данной разработке используется собственная, упрощенная реализация графа.

В различных программных продуктах для программирования графики в режиме реального времени и создания игр трансформации традиционно производятся раз в единицу времени (частота обычно измеряется в кадрах в секунду) в специальном методе (например, в среде разработки для движка Unity таковым является метод Update(), вызываемый раз в кадр у всех объектов, являющихся производными от базового класса MonoBehaviour [6]).

Компоненты в данной разработке, представляющие графические примитивы, работают по такому же принципу. У них есть внутреннее состояние, которое можно менять либо вручную, либо путём вызова метода Update(), который имеется у всех компонентов, являющихся производными от основного класса графа сцены – GLObject.

## Описание алгоритма и функционирования программы

### Описание алгоритма программы

Ниже приводится краткая последовательность действий, выполняемых программой в процессе работы в автономном режиме:

* Пользователь запускает .jar-файл, содержащий программу;
* Программа создаёт окно, содержащее экземпляр компонента OpenGLViewer и вспомогательное меню для работы с компонентом, а также инициализирует пустой граф сцены;
* Пользователь может либо загрузить готовый граф из файла через меню, либо начать добавлять новые объекты в граф;
  + Во время добавления через меню каждого нового объекта пользователю будет предложено указать его начальные координаты, размер, цвет и другие параметры в зависимости от типа объекта;
* Пользователь может также настроить отображение осей координат (включено/выключено), отображение отладочного текста (включено/выключено) и состояние отображения сцены (рендеринг включен/выключен);
* Во время просмотра пользователь может пользоваться возможностью обзора сцены с помощью клавиш-стрелок, Shift и Ctrl для перемещения по всем осям координат и мышки для изменения угла обзора или приближения/отдаления сцены;
* После окончания работы со сценой пользователь может сохранить текущую сцену в файл (происходит сериализация графа сцены в формат json).

### Описание функционирования программы

Структура набора компонентов состоит из нескольких пакетов с классами. Основные из них:

* **objects**, содержащий компоненты, представляющие собой графические примитивы;
* **viewer**, содержащий компонент OpenGLViewer для просмотра сцен;
* **window**, содержащий компонент окна для запуска набора в автономном режиме.

### Пакет objects и класс GLObject

Основной класс пакета – **GLObject**. Это абстрактный класс, представляющий узел графа сцены, а также графический примитив, который будет отрисован. Все примитивы в пакете наследуются от данного класса. GLObject содержит ссылку на список дочерних объектов графа, тоже являющихся экземплярами GLObject. Объекту можно также присвоить идентификатор и текстуру.

GLObject содержит абстрактные методы draw(), который отвечает за отрисовку объекта на экране посредством команд OpenGL, и update(), который изменяет состояние объекта раз в единицу времени.

Для удобства отрисовки GLObject предоставляет метод drawTree(), который отрисовывает объект, вызывает метод update() и рекурсивно повторяет процедуру для всех потомков объекта, таким образом производя рендеринг всего дерева.

Помимо реализации метода, каждый дочерний класс в пакете содержит информацию о состоянии объекта (размер, цвет и т.д.) в виде дополнительных полей.

### Пакет viewer и класс OpenGLViewer

**OpenGLViewer** – основной компонент набора. Он обеспечивает подключение различных возможностей OpenGL (свет, текстурирование и т.д.), отрисовку, поддерживает вид на сцену (позицию в пространстве и угол поворота камеры), предоставляет интерфейс для изменения этого вида с помощью пользовательских команд, контролирует состояние рендеринга, показа отладочного текста и осей координат при необходимости.

OpenGLViewer содержит ссылку на корневую вершину графа сцены и раз в кадр производит обход этого графа в глубину, производя последовательную отрисовку объектов графа.

В пакете viewer также присутствует компонент **GLViewerCanvas** – обертка над OpenGLViewer, содержащая обработчики событий нажатия на клавиатуру и движения мышью.

### Пакет window и класс OpenGLTestFrame

Пакет window содержит единственный компонент **OpenGLTestFrame**, представляющий собой окно для работы с набором компонентов в автономном режиме.

В состав компонента входят экземпляр компонента GLViewerCanvas (см. п. 3.3.2.2) и меню для работы со сценой (возможности меню представлены в п. 3.3.1).

Экземпляр компонента создаётся по умолчанию в случае запуска набора компонентов как отдельного jar-приложения.

## Описание и обоснование выбора метода организации входных и выходных данных

### Описание и обоснование выбора метода организации входных данных

В качестве входных данных для компонента OpenGLViewer используется объект графа сцены (класс GLObject или производные от него). В случае работы набора компонентов как отдельного приложения (автономный режим) граф может либо создаваться пользователем в окне OpenGLTestFrame (см. 3.3.2.3) с помощью меню добавления объектов, либо как сериализованный в json объект GLObject (опция меню окна).

В случае подключения компонентов в сторонние библиотеки или приложения граф сцены задаётся в компоненте OpenGLViewer через геттеры и сеттеры.

Обоснование выбора графа сцены описано в п. 3.2. данного документа.

### Описание и обоснование выбора метода организации выходных данных

В качестве выходных данных компонент OpenGLViewer выводит отрисованное изображение сцены в окно компонента.

Если пользователь работает с набором компонентов в режиме отдельного приложения, есть возможность сохранить созданную пользователем сцену в файл в формате json (сериализовать граф сцены). Данное действие можно выполнить с помощью меню компонента OpenGLTestFrame.

## Описание и обоснование выбора технических и программных средств

Языком разработки компонентов является Java, для работы с OpenGL используется библиотека Java OpenGL (JOGL), для сериализации компонентов – библиотека gson [7].

Были обозначены следующие минимальные требования к аппаратному обеспечению, необходимые для работы программы:

* + Минимальная тактовая частота процессора – 1 Ггц;
  + Минимальный объем ОЗУ – 512 Мб;
  + Минимальное свободное место на жёстком диске для хранения программы – 10 Мб;
  + Монитор с минимальным разрешением 800х600;
  + Видеокарта с минимальным размером видеопамяти 512 Мб;
  + Клавиатура и мышь.

Минимальные требования к программному обеспечению, необходимые для работы программы:

* ОС Windows XP или выше;
* Java SE Runtime Environment 8 или выше;
* Драйвер видеокарты с поддержкой OpenGL 2.1 или выше.

# ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

* 1. Предполагаемая потребностьНабор компонентов потенциально упростит просмотр, отладку и нахождение ошибок при построении 3D-моделей и таким образом может использоваться программистами в области компьютерной графики и 3D-моделистами.
  2. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогамиНа момент начала разработки аналогов открытого отечественного и зарубежного разрабатываемого программного обеспечения не нашлось.

# ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ

1. Understanding and Implementing Scene Graphs [Electronic resource]. // GameDev.net [Official website]. URL: <http://archive.gamedev.net/archive/reference/programming/features/scenegraph/index.html> (accessed: 14.04.2019)

2. OpenGL 2.1 Reference Pages [Electronic resource]. // The Khronos Group Inc [Official website]. URL: <https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/> (accessed: 14.04.2019)

3. JOGL – Java Binding for the OpenGL API [Electronic resource]. // JogAmp.org - Java graphics, audio, media and processing libraries exposing OpenGL, OpenCL, OpenAL and OpenMAX [Official website]. URL: <http://jogamp.org/jogl/www/> (accessed: 14.04.2019)

4. Advantages of component-based development [Electronic resource]. // ResearchGate [Official website]. URL: <https://www.researchgate.net/figure/Advantages-of-component-based-development_tbl1_220588371> (accessed: 14.04.2019)

5. JavaBeans Spec [Electronic resource]. // Oracle [Official website]. URL: <https://www.oracle.com/technetwork/articles/javaee/spec-136004.html> (accessed: 14.04.2019)

6. Scripting API: MonoBehaviour.Update() [Electronic resource]. // Unity [Official website]. URL: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html> (accessed: 14.04.2019)

7. A Java serialization/deserialization library to convert Java objects into JSON and back [Electronic resource]. // Github [Official website]. URL: <https://github.com/google/gson> (accessed: 14.04.2019)

8. gson/RuntimeTypeAdapterFactory.java [Electronic resource]. // Github [Official website]. URL: <https://github.com/google/gson/blob/master/extras/src/main/java/com/google/gson/typeadapters/RuntimeTypeAdapterFactory.java> (accessed: 14.04.2019)

# Приложение 1

**Описание и функциональное назначение классов**

**Пакет** ru.dansstuff.simpleopengl

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| Main | Статический класс, содержащий единственный метод main(), запускающий программу в автономном режиме. |

**Пакет** ru.dansstuff.simpleopengl.math

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| Vec3 | Класс, представляющий трехмерный числовой вектор. |
| Vec4 | Класс, представляющий четырехмерный числовой вектор. |

**Пакет** ru.dansstuff.simpleopengl.misc.gson

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| RuntimeTypeAdapterFactory | Класс-дополнение библиотеки gson для сериализации объектов [8] |

**Пакет** ru.dansstuff.simpleopengl.misc.helpers

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| ObjectCreationFrameFactory | Класс для создания оконных объектов конфигурации новых примитивов |
| SceneFileHelper | Статический класс для сериализации графов сцены |

**Пакет** ru.dansstuff.simpleopengl.objects

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| Box | Класс, представляющий примитив «Параллелепипед» |
| Cylinder | Класс, представляющий примитив «Цилиндр» |
| DirectionalLight | Класс, представляющий источник света |
| EmptyObject | Класс, представляющий пустой примитив для привязки сцены |
| GLObject | Базовый класс примитивов и вершин графа, от которого наследуются все классы-примитивы |
| Line | Класс, представляющий примитив «Линия» |
| OpenGLColor | Класс, представляющий цвет в палитре RGB |
| Sphere | Класс, представляющий примитив «Сфера» |
| Triangle | Класс, представляющий примитив «Треугольник» |

**Пакет** ru.dansstuff.simpleopengl.objects.windows

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| BoxFrame | Класс, представляющий окно создания примитива «Параллелепипед» |
| CylinderFrame | Класс, представляющий окно создания примитива «Цилиндр» |
| DirectionalLightFrame | Класс, представляющий окно создания источника света |
| EmptyObjectFrame | Класс, представляющий окно создания пустого примитива |
| LineFrame | Класс, представляющий окно создания примитива «Линия» |
| SphereFrame | Класс, представляющий окно создания примитива «Сфера» |
| TriangleFrame | Класс, представляющий окно создания примитива «Треугольник» |
| TypeBaseFrame | Абстрактный класс, от которого наследуются все классы пакета |

**Пакет** ru.dansstuff.simpleopengl.operations

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| OpenGLOperation | Интерфейс, представляющий трансформацию сцены или объекта |
| Rotation | Класс, представляющий трансформацию поворота |
| Translation | Класс, представляющий трансформацию сдвига |

**Пакет** ru.dansstuff.simpleopengl.viewer

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| GLViewerCanvas | Класс-обёртка над OpenGLViewer для привязки обработчиков событий |
| OpenGLViewer | Главный компонент рендеринга сцены |

**Пакет** ru.dansstuff.simpleopengl.viewer.listeners

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| OpenGLViewerPopupMenu | Контекстное меню, отображаемое при нажатии на окно OpenGLViewer правой клавишей мыши |
| OpenGLViewerKeyListener | Обработчик события нажатия на клавишу клавиатуры |
| OpenGLViewerPopupMenu | Обработчик событий, связанных с движением или нажатием на различные клавиши мыши |

**Пакет** ru.dansstuff.simpleopengl.window

|  |  |
| --- | --- |
| **Имя класса** | **Назначение** |
| OpenGLTestFrame | Окно для работы набора компонентов в режиме отдельного приложения |

# Приложение 2

**Описание и функциональное назначение полей/свойств и методов классов**

**Примечание:** в списках методов не описываются методы доступа (геттеры) и модификации (сеттеры). Поля, снабжённые геттерами, описываются вместе с модификатором @G, поля, снабжённые сеттерами, описываются вместе с модификатором @S.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Класс** ru.dansstuff.simpleopengl.**Main** | | | | | | | |
| **Поля** | | | | | | | |
| Имя | Модификатор доступа | | Тип | | | Описание | |
| width | private static | | int | | | Длина окна OpenGLTestFrame. | |
| height | private static | | int | | | Ширина окна OpenGLTestFrame. | |
| **Методы** | | | | | | | |
| Имя | | Модификатор доступа | | Тип | Параметры | | Описание |
| main | | public static | | void | String[] args | | Запускает программу. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Класс** ru.dansstuff.simpleopengl.math.**Vec3** | | | | | | | |
| **Поля** | | | | | | | |
| Имя | Модификатор доступа | | Тип | | | Описание | |
| @G @S x | private | | float | | | X-компонента вектора. | |
| @G @S y | private | | float | | | Y-компонента вектора. | |
| @G @S z | private | | float | | | Z-компонента вектора. | |
| **Методы** | | | | | | | |
| Имя | | Модификатор доступа | | Тип | Параметры | | Описание |
| Vec3 | | public | | Vec3 | float x, float y, float z | | Конструктор класса. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Класс** ru.dansstuff.simpleopengl.math.**Vec4** | | | | | | | |
| **Поля** | | | | | | | |
| Имя | Модификатор доступа | | Тип | | | Описание | |
| @G @S x | private | | float | | | X-компонента вектора. | |
| @G @S y | private | | float | | | Y-компонента вектора. | |
| @G @S z | private | | float | | | Z-компонента вектора. | |
| @G @S w | private | | float | | | W-компонента вектора. | |
| **Методы** | | | | | | | |
| Имя | | Модификатор доступа | | Тип | Параметры | | Описание |
| Vec4 | | public | | Vec4 | float x, float y, float z, float w | | Конструктор класса. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Класс** ru.dansstuff.simpleopengl.misc.helpers.**ObjectCreationFrameFactory** | | | | | | | |
| **Поля** | | | | | | | |
| Имя | Модификатор доступа | | Тип | | | Описание | |
| frameMap | private | | Map<Class, Class> | | | Отображение для получения объектов окон создания примитивов. | |
| **Методы** | | | | | | | |
| Имя | | Модификатор доступа | | Тип | Параметры | | Описание |
| getFrame | | public static | | JFrame | Class clazz, OpenGLTestFrame parent | | Возвращает экземпляр окна создания примитива в зависимости от переданного класса примитива. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Класс** ru.dansstuff.simpleopengl.misc.helpers.**SceneFileHelper** | | | | | | | |
| **Поля** | | | | | | | |
| Имя | Модификатор доступа | | Тип | | | Описание | |
| rFactory | private static | | RuntimeTypeAdapterFactory<GLObject> | | | Экземпляр RuntimeTypeAdapterFactory для сериализации и десериализации полиморфных типов. | |
| gson | private static | | Gson | | | Сериализатор объектов. | |
| **Методы** | | | | | | | |
| Имя | | Модификатор доступа | | Тип | Параметры | | Описание |
| readScene | | public static | | GLObject | File file | | Десериализует файл в объект графа сцены. |
| writeScene | | public static | | void | GLObject root, File file | | Сериализует объект графа сцены в указанный файл. |
| getSceneJson | | public static | | string | GLObject root | | Получает строковую репрезентацию сериализованного графа сцены |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Класс** ru.dansstuff.simpleopengl.misc.objects.windows.**TypeBaseFrame**  Примечание: все производные классы данного класса, содержащиеся в пакете ru.dansstuff.simpleopengl.misc.objects.windows, не описываются в данном Приложении, поскольку в них присутствует только конфигурация расположения графических виджетов, связанных с полями тех примитивов, которые должны быть созданы. | | | | | | | |
| **Поля** | | | | | | | |
| Имя | Модификатор доступа | | Тип | | | Описание | |
| @G @S parent | protected | | OpenGLTestFrame | | | Экземпляр окна OpenGLTestFrame для добавления объектов. | |
| **Методы** | | | | | | | |
| Имя | | Модификатор доступа | | Тип | Параметры | | Описание |
| createObject | | protected abstract | | void |  | | Добавляет объект с созданной в окне конфигурацией в граф сцены. |
| getNum | | protected | | int | JTextField field | | Получает численный пользовательский ввод из графических виджетов, наследующихся от компонента JTextField. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Класс** ru.dansstuff.simpleopengl.objects.**GLObject** | | | | | | | |
| **Поля** | | | | | | | |
| Имя | | Модификатор доступа | | Тип | | Описание | |
| @G @S id | | protected | | String | | Идентификатор объекта. | |
| @G @S children | | protected | | List<GLObject> | | Список потомков объекта. | |
| @G type | | protected | | String | | Тип объекта. | |
| @G @S textureFile | | protected | | String | | Путь к текстуре объекта в файловой системе. | |
| @G @S texture | | protected | | com.jogamp.opengl.util.texture.Texture | | Текстура объекта. | |
| @G frameClass | | protected | | Class | | Класс окна создания объекта. | |
| **Методы** | | | | | | | |
| Имя | Модификатор доступа | | Тип | | Параметры | | Описание |
| draw | public abstract | | void | | com.jogamp.opengl.GL2 gl | | Отрисовывает объект. |
| update | public abstract | | void | |  | | Обновляет состояние объекта. |
| resolveTexturesForTree | public | | void | | Map<String, Texture> textureMap | | Загружает текстуры из файлов и присваивает их всем объектам графа сцены, которые являются потомками данного объекта. |
| getTextureFromFile | private | | com.jogamp.opengl.util.texture.Texture | | File file | | Получает текстуру из файла |
| resolveTexture | public | | void | | Map<String, Texture> textureMap | | Назначает уже существующую текстуру данному объекту, либо получает ее из файла |
| addChild | public | | void | | GLObject child | | Добавляет потомка данного объекта в граф сцены |
| drawTree | public | | void | | com.jogamp.opengl.GL2 gl | | Отрисовывает граф сцены, начиная с данного объекта |
| getObjectsCount | public | | int | |  | | Возвращает количество объектов в графе сцены, начиная с данного объекта |
| getTreeAsList | public | | List<GLObject> | |  | | Возвращает репрезентацию графа сцены в виде одномерного списка |
| clear | public | | void | |  | | Очищает граф сцены |
| getObjectTypes | public static | | Set<Class> | |  | | Возвращает все зарегистрированные типы примитивов |

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |