Национальный исследовательский университет – Высшая школа экономики

Факультет бизнес-информатики, отделение программной инженерии

УТВЕРЖДЕНО

Заведующий кафедрой «Управление разработкой программного обеспечения»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Авдошин С.М./

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2012 г.

**Компонентная модель с декларативным описанием составных типов: Парсеры**

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Подп. и дата |  | | Инв. № дубл. |  | | Взам. инв. № |  | | Подп. и дата |  | | Инв. № подп. |  | | |  | | --- | | Руководитель работы  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Гринкруг Е.М./  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2012 г. | | Исполнитель: студент группы 271ПИ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Дубов М.С. /  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2012 г. |   2012 |

Национальный исследовательский университет – Высшая школа экономики

Факультет бизнес-информатики, отделение программной инженерии

УТВЕРЖДЕНО

**Компонентная модель с декларативным описанием составных типов: парсеры**

Пояснительная записка

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Подп. и дата |  | | Инв. № дубл. |  | | Взам. инв. № |  | | Подп. и дата |  | | Инв. № подп. |  | | Листов 13  2012 |

# Содержание

[Содержание 2](#_Toc325413664)

[1. Введение 3](#_Toc325413665)

[1.1. Общие сведения о программе 3](#_Toc325413666)

[1.2. Основания для разработки 3](#_Toc325413667)

[1.2.1. Цель разработки 3](#_Toc325413668)

[1.2.2. Заказчик проекта 3](#_Toc325413669)

[2. Назначение разработки 3](#_Toc325413670)

[2.1. Назначение программы 3](#_Toc325413671)

[2.2. Область применения программы 3](#_Toc325413672)

[3. Технические характеристики 4](#_Toc325413673)

[3.1. Постановка задачи 4](#_Toc325413674)

[3.2. Используемые алгоритмы 4](#_Toc325413675)

[3.3. Метод организации входных и выходных данных 5](#_Toc325413676)

[3.3.1. Входные данные 5](#_Toc325413677)

[3.3.2. Выходные данные 5](#_Toc325413678)

[3.4. Состав технических и программных средств 5](#_Toc325413679)

[3.4.1. Технические средства 5](#_Toc325413680)

[3.4.2. Программные средства 5](#_Toc325413681)

[4. Ожидаемые технико-экономические показатели 6](#_Toc325413682)

[5. Источники, используемые при разработке 6](#_Toc325413683)

[6. Приложение А. Описание и функциональное назначение классов и структур 7](#_Toc325413684)

[6.1. Библиотека парсеров 7](#_Toc325413685)

[6.2. Библиотека средств генерации кода 7](#_Toc325413686)

[6.3. Стандартные узлы и типы данных VRML/X3D 7](#_Toc325413687)

[7. Приложение Б. Описание и функциональное назначение методов, полей и свойств 8](#_Toc325413688)

[7.1. Библиотека парсеров 8](#_Toc325413689)

[7.1.1. Абстрактный класс Parser 8](#_Toc325413690)

[7.1.2. Класс VRMLParser 10](#_Toc325413691)

[7.1.3. Класс X3DParser 11](#_Toc325413692)

[7.2. Библиотека средств генерации кода 12](#_Toc325413693)

[7.2.1. Абстрактный класс CodeGenerator 12](#_Toc325413694)

[7.2.2. Класс VRMLCodeGenerator 12](#_Toc325413695)

[7.2.3. Класс X3DCodeGenerator 13](#_Toc325413696)

[7.3. Стандартные узлы VRML/X3D 13](#_Toc325413697)

[7.3.1. Абстрактный базовый класс Node 13](#_Toc325413698)

[7.3.2. Стандартные и сторонние классы-узлы 13](#_Toc325413699)

# Введение

## Общие сведения о программе

**Наименование программы**: Библиотека парсеров декларативного описания компонентных моделей.

Библиотека состоит из двух компонент:

* Первая компонента – набор средств для синтаксического анализа (парсинга) декларативного описания компонентных моделей;
* Вторая компонента – набор средств для генерации декларативного описания компонентных моделей.

## Основания для разработки

## Цель разработки

Разработка осуществляется соответственно следующим документам:

* «Список тем курсовых работ студентов отделения программной инженерии факультета бизнес-информатики – 2 курс».
* Техническое задание

Целью разработки является создание библиотеки средств парсинга и кодогенерации для ее внедрения в программу визуализации архитектуры компонентных моделей на основе их декларативного описания.

## Заказчик проекта

Заказчиком проекта является НИУ-ВШЭ, отделение программной инженерии факультета бизнес-информатики, кафедра «Управление разработкой программного обеспечения».

# Назначение разработки

## 2.1. Назначение программы

Программный комплекс предназначен для построения компонентных моделей на основе их описания на одном из поддерживаемых декларативных языков (VRML/X3D), а также для генерации декларативного описания уже существующих моделей. Библиотека предназначена для использования сторонними разработчиками при разработке ими других приложений.

## 2.2. Область применения программы

* Построение редакторов компонентных моделей;
* Анализ ошибок в коде декларативного описания (например, в специальных текстовых редакторах)
* Программы визуализации компонентных моделей (например, в виде 3D-сцен);
* Конвертирование между представлениями одной и той же модели на разных декларативных языках.

# Технические характеристики

## 3.1. Постановка задачи

Библиотека парсеров должна осуществлять синтаксический анализ подаваемых ей на вход исходных файлов формата VRML или X3D, проверять их на наличие лексических, синтаксических или семантических ошибок и в случае их отсутствия строить модель прочитанной сцены – направленный ациклический граф. Граф строится в виде специального внутреннего представления с помощью определенных в библиотеке Java-классов, представляющих его узлы; это представление затем может быть использовано для траверсирования этого графа, и, таким образом, для использования результата синтаксического анализа в других приложениях.

Библиотека средств генерации кода должна выполнять обратную операцию: на основе внутреннего представления графа генерировать код на языках VRML или X3D. Таким образом, вместе с парсерами эта часть библиотеки должна позволять осуществлять конвертацию из классического VRML-формата в более новый XML-подобный формат X3D и наоборот.

Предполагается внедрение библиотеки в программу-редактор архитектуры компонентных моделей. Описанные выше функции должны обеспечивать загрузку моделей из файлов (содержащих их декларативное описание) и последующее сохранение этих моделей в файл.

## 3.2. Используемые алгоритмы

Парсер языка VRML строится на основе формальной грамматики языка (данная грамматика является контекстно-свободной и описана в стандарте ISO [5]) с помощью *метода рекурсивного спуска* (*recursive-descent parsing* [1]). В таком парсере каждому из продукций грамматики в коде соответствует специальный метод. Это позволяет довольно строго организовать код парсера, повысить его читаемость и поддерживаемость, однако приводит к большому числу рекурсивных вызовов при анализе исходных файлов с высоким уровнем вложенности узлов.

Важно отметить, что грамматика языка VRML является *неоднозначной* (*ambiguous*): на основе ее одной невозможно осуществлять анализ кода, так как, например, на момент считывания значений полей узлов тип их неизвестен. Для решения этой проблемы активно используется реализованный в библиотеках Java механизм *рефлексии* ([2, 3]): во время считывания описания определенного узла исследуется код соответствующего этому узлу Java-класса (описанного в библиотеке или сторонним разработчиком) и определяется, значения каких типов могут храниться в его полях.

Отличительной деталью парсера является реализованная в нем возможность довольно детальной диагностики ошибок в исходных файлах. Так, лексические ошибки распознаются при неверном написании названий полей и узлов; использование рефлексии при этом позволяет получать список идентификаторов, возможных на месте неверного, и предлагать программисту на основе этого списка наиболее вероятное исправление. Синтаксические ошибки имеют место, например, при отсутствии в нужных местах открывающих/закрывающих фигурных скобок. Встречая такие ошибки, парсер способен восстанавливаться, используя так называемый «*режим паники*» (*panic-mode recovery* [1]), что позволяет ему продолжать чтение исходного файла и, таким образом, сообщать о как можно большем числе ошибок за один проход.

Язык X3D является XML-подобным языком ([6]), поэтому его парсер строится с помощью методологии *SAX* (*Simple API for XML* [4]). Этот парсер является событийным: в основе его работы – отслеживание ограниченного ряда событий (таких, как наличие открывающего тега, атрибута и др.) в ходе прохода по файлу и соответствующая их обработка. Аналогично VRML-парсеру этот парсер также использует рефлексию.

Оба парсера работают за время *O(n)*, где *n* – число символов во входном файле. Для лексического анализа исходных файлов (разбиения их на лексемы) используется реализованный в библиотеке Java лексический анализатор *StreamTokenizer*.

Генерация кода осуществляется в ходе *траверсирования* графа ([8]): оно основано на проходе по всем дочерним узлам каждого из узлов графа и реализовано рекурсивно.

## 3.3. Метод организации входных и выходных данных

## 3.3.1. Входные данные

Входными данными для парсеров VRML и X3D являются текстовые файлы, содержащие декларативное описание сцены на этих языках в соответствии со стандартом ([5] и [6]). Число узлов, которые могут быть описаны в этих файлах, также ограничено стандартом, однако может быть расширено программистом с помощью реализации соответствующих Java-классов и регистрации их перед использованием парсера.

## 3.3.2. Выходные данные

Выходные данные парсера – направленный ациклический граф сцены, представленный в виде массива корневых узлов. Каждый же узел представляет собой экземпляр специального класса, соответствующего определенному типу узла и являющегося наследником определенного в библиотеке класса *Node*. Каждый из таких классов должен быть реализован в соответствии со стандартом JavaBeans [7], что позволяет выполнять их *интроспекцию* ([3]) с помощью механизма рефлексии ([2]). Так, ссылки на дочерние узлы содержатся в getter’ах узлов и легко могут быть получены в ходе обхода графа.

Выходные данные парсеров являются входными данными для средства кодогенерации; входные для парсеров, соответственно, выходными для генераторов кода.

## 3.4. Состав технических и программных средств

## 3.4.1. Технические средства

Основным средством распространения программного комплекса является веб-сервис для хостинга открытых проектов GitHub (<https://github.com/msdubov/Component-model>), что продиктовано возможностью использования исходного кода в других программных продуктах. Альтернативным способом распространения программного комплекса является распространение на носителях типа CD-ROM.

## 3.4.2. Программные средства

Для работы библиотеки необходима реализация виртуальной машины Java версии не ниже 6 (например, Java Runtime Environment). При использовании библиотека в ходе разработки других программных продуктов необходимо также наличие средств Java Development Kit.

# Ожидаемые технико-экономические показатели

Представленный в библиотеке VRML-парсер является одним из наиболее развитых с точки зрения диагностики ошибок в исходных файлах. Синтаксический анализатор способен распознавать лексические, синтаксические и семантические ошибки в исходном коде, восстанавливаться при наличии таких ошибок и продолжать анализ исходного текста, обрабатывая, таким образом, максимально возможное число ошибок за один проход. Это позволяет успешно использовать его при создании интегрированных средств разработки на VRML в составе текстовых редакторов с возможностью подчеркивания ошибок в исходном коде.

В настоящее время на смену стандарту трехмерной векторной графики VRML приходит более современный стандарт X3D, вводящий, в том числе, и новый XML-подобный формат кодирования сцен. С этим может быть связана необходимость конвертировать уже существующие декларативные описания сцен из классического VRML-формата в новый XML-формат. Наличие в библиотеке соответствующих парсеров и средств кодогенерации позволяет выполнять как эту, так и обратную к ней конвертацию.

# Источники, используемые при разработке

1. A. V. Aho, M. S. Lam, R. Sethi and J. D. Ullman, Compilers: principles, techniques, and tools, 2nd ed. MA: Prentice Hall, 2006.
2. C. S. Horstmann and G. Cornell, Core Java, 8th ed., vol. 1: Fundamentals. MA: Prentice Hall, 2007.
3. C. S. Horstmann and G. Cornell, Core Java, 8th ed., vol. 2: Advanced features. MA: Prentice Hall, 2008.
4. <http://en.wikipedia.org/wiki/Simple_API_for_XML>
5. ISO/IEC 14772-1:1997 and ISO/IEC 14772-2:2004 — Virtual Reality Modeling Language (VRML).
6. ISO/IEC 19775 – X3D.
7. Sun Microsystems, JavaBeans Specification v1.0.1, July 1997. <http://java.sun.com/products/javabeans/docs/spec.html>
8. R. Sedgewick, Algorithms in Java, 4th ed., CA: Addison-Wesley Educational Publishers Inc., 2010.

# Приложение А. Описание и функциональное назначение классов и структур

## 6.1. Библиотека парсеров

|  |  |
| --- | --- |
| **Имя класса/структуры** | **Описание** |
| **Parser** | Абстрактный базовый класс, реализующий общую функциональность парсеров VRML и X3D. |
| **VRMLParser** | Парсер VRML. |
| **X3DParser** | Парсер X3D. |
| **ParsingError** | Абстрактный класс-наследник стандартного класса Error, реализующий некоторую общую логику обработки ошибок синтасического анализа. |
| **SyntaxError** | Класс-наследник *ParsingError*, экземпляры которого содержат информацию о синтаксической ошибке в исходном файле. |
| **LexicalError** | Класс-наследник *ParsingError*, экземпляры которого содержат информацию о лексической ошибке в исходном файле. |
| **TypeMismatchError** | Класс-наследник *ParsingError*, экземпляры которого содержат информацию об ошибке несоответствия типов в исходном файле. |
| **Warning** | Класс-наследник *ParsingError*, экземпляры которого содержат информацию о предупреждениях (некритичных ошибках) в исходном файле. |

## 6.2. Библиотека средств генерации кода

|  |  |
| --- | --- |
| **Имя класса/структуры** | **Описание** |
| **CodeGenerator** | Абстрактный базовый класс, реализующий общую функциональность генераторов кода VRML и X3D. |
| **VRMLCodeGenerator** | Генератор кода VRML. |
| **X3DCodeGenerator** | Генератор кода X3D. |

## 6.3. Стандартные узлы и типы данных VRML/X3D

|  |  |
| --- | --- |
| **Имя класса/структуры** | **Описание** |
| **VRMLType** | Абстрактный базовый класс для *Node* и *ValueType.* |
| **Node** | Абстрактный базовый класс для всех классов, представляющих узлы VRML/X3D. |
| **Appearance** | Узел, отвечающий за параметры рендеринга объекта. |
| **Box** | Узел, представляющий прямоугольный параллелепипед. |
| **Geometry** | Абстрактный тип узла, представляющего некоторый геометрический объект. |
| **Group** | Узел, группирующий другие объекты |
| **Material** | Узел, отвечающий за текстуру объекта. |
| **Shape** | Узел, представляющий некоторую геометрическую фигуру. |
| **Sphere** | Узел, представляющий сферу. |
| **Text** | Узел, представляющий текст. |
| **ValueType** | Абстрактный базовый класс для типов значений полей узлов VRML/X3D. |
| **SFBool** | Стандартный булев тип. |
| **SFFloat** | Стандартный вещественный тип. |
| **SFInt32** | Стандартный целочисленный тип. |
| **SFString** | Стандартный строковый тип. |
| **SFColor** | Тип “цвет”, представляет собой тройку значений SFFloat. |
| **MFType** | Абстрактный тип, поддерживающий множественные значения. |
| **MFValueType** | Абстрактный множественный тип для типов-значений. |
| **MFBool** | Множественный тип для булевых значений. |
| **MFFloat** | Множественный тип для вещественных значений. |
| **MFInt32** | Множественный тип для целочисленных значений. |
| **MFNode** | Множественный тип для типов-узлов. |
| **MFString** | Множественный тип для строковых значений. |

# Приложение Б. Описание и функциональное назначение методов, полей и свойств

## 7.1. Библиотека парсеров

## 7.1.1. Абстрактный класс Parser

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя** | **Модифика- торы** | **Тип** | **Аргументы** | **Описание** |
| **Методы** | | | | |
| **parse** | public | ArrayList  <Node> | InputStream  Reader | Принимает на вход строковый поток и возвращает граф сцены, либо null в случае наличия ошибок в исходном тексте. |
| **setUp**  **Tokenizer** | protected | void | - | Настраивает лексический анализатор. |
| **init** | protected abstract | void | - | Инициализирует парсер, читает первую лексему из потока. |
| **parseScene** | protected abstract | void | - | Формирует граф сцены на основе синтаксического анализа входного файла. |
| **parseChild**  **Node** | public abstract | Node | - | Читает из входного потока следующий узел и возвращает экземпляр соответствующего класса. |
| **parseValue**  **Type** | protected | Object | Class<?> | Читает из входного потока значение заданного типа. |
| **tokenizer** | public | Stream Tokenizer | - | Возвращает объект, представляющий лексический анализатор. |
| **lookahead** | public | boolean | String | Сравнивает текущую лексему в потоке с аргументом. |
| **lookahead** | public | String | - | Возвращает текущую лексему в потоке. |
| **nextToken** | public abstract | boolean | - | Считывает следующую лексему из потока. |
| **match** | public | boolean | String | Сопоставляет текущую лексему с аргументом и генерирует ошибку в случае несоответствия. |
| **tryMatch** | public | boolean | String | Сопоставляет текущую лексему с аргументом; запоминает возможную ошибку в случае несоответствия. |
| **registerError** | public | boolean | Error | Регистрирует ошибку парсинга. |
| **getParsing**  **Errors** | public | ArrayList  <Error> | - | После неудачной попытки парсинга возвращает сформированный список ошибок. |
| **classFor**  **NodeName** | protected | Class<?> | String | Осуществляет поиск класса-узла по имени в одном из зарегистрированных пакетов с классами-узлами. |
| **create**  **Instance** | protected | Node | String | Возвращает объект класса-узла по его имени. |
| **registerNodePackage** | public | void | String | Регистрирует пакет с классами-узлами. |
| **Поля** | | | | |
| **tokenizer** | protected | Stream  Tokenizer |  | Лексический анализатор. |
| **sceneGraph** | protected | ArrayList  <Node> |  | Граф сцены, представляемый в виде списка корневых узлов. |
| **parsing**  **Errors** | protected | ArrayList  <Error> |  | Список ошибок парсинга. |
| **possibleError** | protected | Error |  | Возможная ошибка, зарегистрированная в tryXxx методе. |
| **nodePackages** | protected | ArrayList  <String> |  | Список зарегистрированных пакетов с классами-узлами. |

## 7.1.2. Класс VRMLParser

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя** | **Модифика- торы** | **Тип** | **Аргументы** | **Описание** |
| **Методы** | | | | |
| **setUp**  **Tokenizer** | protected | void | - | Настраивает лексический анализатор. |
| **init** | protected | void | - | Инициализирует парсер, читает первую лексему из потока. |
| **parseScene** | protected | void | - | Формирует граф сцены на основе синтаксического анализа входного файла. |
| **parseXxx**  **[…]** | public | boolean | - | Один из методов, соответствующих продукциям грамматики VRML. |
| **nextToken** | public | boolean | - | Считывает следующую лексему из потока. |
| **lookahead**  **IsId** | private | boolean | - | Определяет, является ли текущая лексема идентификатором. |
| **lookahead IsFieldName** | private | boolean | - | Определяет, является ли текущая лексема названием одного из полей текущего узла. |
| **tryMatch**  **FieldId** | private | boolean | - | Осуществляет попытку сопоставления текущей лексемы с названием поля текущего узла. |
| **tryMatch**  **TypeId** | private | boolean | - | Осуществляет попытку сопоставления текущей лексемы с именем узла. |
| **panicMode**  **Recovery** | private | boolean | - | Восстановление после ошибок парсинга в «режиме паники». |
| **instantiate**  **Node** | private | boolean | - | Инстанциирует узел по его типу. |
| **instantiate**  **NodeById** | private | boolean | - | Инстанциирует узел по его ID с помощью хэш-таблицы узлов. |
| **addRootNode** | private | boolean | - | Добавляет текущий узел на первый уровень графа |
| **matchField**  **ValueAnd**  **SetField** | private | boolean | - | Считывает значение поля и записывает его в объект-узел. |
| **initFields** | private | void | - | Инициализирует private-поля класса. |
| **Поля** | | | | |
| **defNodes**  **Table** | private | HashMap  <String, Node> |  | Хэш-таблица DEF-узлов (узлов с ID). |
| **lookahead** | protected | String |  | Текущая лексема. |
| **currentId** | private | String |  | Текущий ID. |
| **currentType** | private | String |  | Тип текущего узла. |
| **currentNodes** | private | Stack<Node> |  | Стек обрабатываемых узлов. |
| **currentField** | private | Stack  <String> |  | Стек считываемых полей. |

## 7.1.3. Класс X3DParser

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя** | **Модифика- торы** | **Тип** | **Аргументы** | **Описание** |
| **Методы** | | | | |
| **setUp**  **Tokenizer** | protected | void | - | Настраивает лексический анализатор. |
| **init** | protected | void | - | Инициализирует парсер, читает первую лексему из потока. |
| **parseScene** | protected | void | - | Формирует граф сцены на основе синтаксического анализа входного файла. |
| **parseXML** | private | void | - | Осуществляет чтение XML и вызов обработчиков SAX-событий. |
| **openingTag** | private | void | String | Обработчик SAX-события «Открывающий тег» |
| **closingTag** | private | void | String | Обработчик SAX-события «Закрывающий тег» |
| **attribute** | private | void | String | Обработчик SAX-события «Атрибут» |
| **textNode** | private | void | String | Обработчик SAX-события «Текстовый узел» |
| **nextToken** | public | boolean | - | Считывает следующую лексему из потока. |
| **match**  **AttributeId** | private | boolean | - | Считывает текущую лексему, которая должна быть идентификатором. |
| **matchField**  **ValueAnd**  **SetField** | private | boolean | String | Считывает значение поля и записывает его в объект-узел. |
| **initFields** | private | void | - | Инициализирует private-поля класса. |
| **Поля** | | | | |
| **defNodes**  **Table** | private | HashMap  <String, Node> |  | Хэш-таблица DEF-узлов (узлов с ID). |
| **lookahead** | protected | String |  | Текущая лексема. |
| **readingTag** | private | boolean |  | Определяет, происходит ли в данный момент считывание тега. |
| **current**  **Attribute** | private | String |  | Имя текущего атрибута. |
| **currentNodes** | private | Stack<Node> |  | Стек обрабатываемых узлов. |
| **currentTags** | private | Stack  <String> |  | Стек считываемых тегов. |
| **fieldValue**  **Name**  **Attributes** | private | Stack  <String> |  | Вспомогательный стек для чтения значений типа MFNode. |
| **fieldValue**  **MFNodes** | private | Stack  <String> |  | Вспомогательный стек для чтения значений типа MFNode. |

## 7.2. Библиотека средств генерации кода

## 7.2.1. Абстрактный класс CodeGenerator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя** | **Модифика- торы** | **Тип** | **Аргументы** | **Описание** |
| **Методы** | | | | |
| **generate** | public abstract | void | ArrayList  <Node>,  PrintStream | Генерирует декларативное описание графа сцены. |
| **VRMLtoX3D** | public static | boolean | InputStream  Reader, PrintStream | Конвертирует код на VRML в код на X3D. |
| **X3DtoVRML** | public static | boolean | - | Конвертирует код на X3D в код на VRML. |

## 7.2.2. Класс VRMLCodeGenerator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя** | **Модифика- торы** | **Тип** | **Аргументы** | **Описание** |
| **Методы** | | | | |
| **generate** | public | void | ArrayList  <Node>,  PrintStream | Генерирует декларативное описание графа сцены. |
| **process** | private | void | Node | Обрабатывает один узел и рекурсивно все его дочерние узлы. |
| **Поля** | | | | |
| **nodes** | private | Stack<Node> |  | Стек обрабатываемых узлов. |
| **output** | private | PrintStream |  | Выходной поток. |
| **defNodes** | private | HashSet  <String> |  | Хэш-таблица встреченных именованных узлов. |

## 7.2.3. Класс X3DCodeGenerator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя** | **Модифика- торы** | **Тип** | **Аргументы** | **Описание** |
| **Методы** | | | | |
| **generate** | public | void | ArrayList  <Node>,  PrintStream | Генерирует декларативное описание графа сцены. |
| **process** | private | void | Node | Обрабатывает один узел и рекурсивно все его дочерние узлы. |
| **Поля** | | | | |
| **nodes** | private | Stack<Node> |  | Стек обрабатываемых узлов. |
| **output** | private | PrintStream |  | Выходной поток. |
| **defNodes** | private | HashSet  <String> |  | Хэш-таблица встреченных именованных узлов. |

## 7.3. Стандартные узлы VRML/X3D

## 7.3.1. Абстрактный базовый класс Node

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя** | **Модифика- торы** | **Тип** | **Аргументы** | **Описание** |
| **Методы** | | | | |
| **setId** | public | void | String | Задает ID узла. |
| **getId** | public | String | - | Возвращает ID узла. |
| **Node** | public | - | - | Публичный конструктор без параметров. |
| **container**  **Field** | public abstract | string |  | Возвращает значение свойства containerField узла (необходимо для X3D-парсинга). |
| **Поля** | | | | |
| **id** | private | String |  | ID узла. |
| **serialVersion**  **UID** | private static final | long |  | Для сериализации узла. |

## 7.3.2. Стандартные и сторонние классы-узлы

Библиотека содержит набор стандартных VRML-узлов, реализованных в виде JavaBeans-компонент. Все эти узлы соответствуют набору требований:

* Реализуют public-конструктор без параметров;
* Обеспечивают доступ к полю xxx на чтение через метод getXxx();
* Обеспечивают доступ к полю xxx на запись через метод setXxx(T value).

Стандартная библиотека может быть расширена пользовательскими узлами, которые также должны быть построены в соответствии со стандартом JavaBeans.