Java Doc

Современный стандарт к документированию программного кода

Требования к документации

- Не документировать очевидные вещи
- Поддерживать документацию в актуальном состоянии
- Описывать входящие параметры, если нужно

Документирование javadoc

- При написании комментариев к кодам Java используют три типа комментариев :

- // однострочный комментарий;
- /* многострочный комментарий */
- /** комментирование документации */

Doc Comments

- Example

```
/**
* Doc comment
*
*/
```

- Regular comment

```
/*
   block comment
*/

// inline comment
```

Документирование javadoc

- При документировании приложения необходима также поддержка документации **кода** программы.
- Если документация и код разделены, то непроизвольно создаются сложности, связанные с необходимостью внесения изменений в соответствующие разделы сопроводительной документации при изменении программного кода.

Документирование javadoc

- Как правило, все существующие среды разработки IDE приложений Java предлагают решение по связыванию кода с документацией в процессе разработки с использованием **javadoc**.
- Для этого необходимо соответствующим образом написать комментарий к коду, т.е. документировать.
- Java комментарии необходимы как для комментирования программы, так и для составления или оформления документации.

Документирование javadoc

- Разработан специальный синтаксис для оформления документации в виде комментариев и инструмент для создания из комментариев документации.
- Этим инструментом является утилита **javadoc**, которая обрабатывая файл с исходным текстом программы, выделяет помеченную документацию из комментариев и связывает с именами соответствующих классов, методов и полей.
- Таким образом, при минимальных усилиях создания комментариев к коду, можно получить хорошую документацию к программе.

Javadoc

- Генератор документации от Sun
- Входит в состав JDK
- Генерирует на базе кодов исходных файлов API документацию в HTML формате
- The **“doc comments”** format is **de facto industry standart** (определяет стандарт для документирования классов Java)

Включение комментариев

Утилита javadoc извлекает информацию о следующих элементах:

- Пакетах
- Классах и интерфейсах, объявленных как **public**
- Методах, объявленных как **public** или **protected**
- Полях, объявленных как **public** или **protected**

Включение комментариев

- Комментарии (javadoc) размещаются **непосредственно перед** элементом, к которому они относятся
- Комментарии имеют вид /** … */ содержат произвольный текст, за которым следует *дескриптор (tag)*
- Дескриптор начинается с символа @, например, **@param**
- Первое предложение текста комментариев представляет краткое описание

Включение и оформление комментариев

В самом тексте комментариев можно использовать элементы языка HTML, например,

- … для выделения текста курсивом
- <code> … </code> для установки моноширинного шрифта
- … </ strong > для выделения текста полужирным шрифтом
- <img> … и даже для вставки рисунков

- НО! Следует избегать заголовков и горизонтальных линий

Structure of a Javadoc Comment

- Помещается в /** */
- Первый параграф … общее описание
- Дескрипторы (тэги, tag, указывающие/классифицирующие описание
  - @ author
  - @version
  - @param
  - @return
  - @throws
  - …

Дескриптор	Применение	Описание
@author	Класс, интерфейс	Автор
@version	Класс, интерфейс	Версия. Не более одного дескриптора на класс
@since	Класс, интерфейс, поле, метод	Указывает, с какой версии доступно
@see	Класс, интерфейс, поле, метод	Ссылка на другое место в документации
@param	Метод	Входной параметр метода
@return	Метод	Описание возвращаемого значения
@exception имя_класса описание	Метод	Описание исключения, которое может быть послано из метода
@throws имя_класса описание	Метод	Описание исключения, которое может быть послано из метода
@deprecated	Класс, интерфейс, поле, метод	Описание устаревших блоков кода
{@link reference}	Класс, интерфейс, поле, метод	Ссылка
{@value}	Статичное поле	Описание значения переменной

Class Car – комментарий к классу

```
3   /**
4    * Это пример простого класса автомобиль
5    *
6    * @author V.C. Sidorik
7    * @version 2.3 21/10/2016
8    */
9   public class Car {
10      private String make;
11      private int year;
12      private int kmage;
13
```

Class Car – комментарий к классу (Javadoc)

Class Car

java.lang.Object
    javadoc.Car

```
public class Car
extends java.lang.Object
```

Это пример простого класса автомобиль

Class Car – комментарий к классу (Javadoc)

by.riit.gui_basic_component.javadoc

Class Car

```
java.lang.Object
  └ by.riit.gui_basic_component.javadoc.Car
```

```
public class Car
extends java.lang.Object
```

This is an example of a simple class car

Version:

2.3 21/10/2016

Author:

V.V. Sidorik

Class Car – комментарий к конструктору

```
14      /**
15       * Конструктор автомобиля заданной фирмы,
16       * года изготовления и пробега
17       *
18       * @param make  make фирма-производитель
19       * @param year  year год изготовления
20       * @param kmage kmage пробег
21       */
22      public Car(String make, int year, int kmage) {
23          this.make = make;
24          this.year = year;
25          this.kmage = kmage;
26      }
```

Class Car – комментарий к конструктору (Javadoc)

Constructor Summary

Constructors

Constructor and Description
`Car(java.lang.String make, int year, int kmage)` Конструктор автомобиля заданной фирмы, года изготовления и пробега

Class Car – комментарий к конструктору (Javadoc)

Constructor Detail

Car

```
public Car(java.lang.String make,
    int year,
    int kmage)
```

Конструктор автомобиля заданной фирмы, года изготовления и пробега

Parameters:

`make` - make фирма-производитель

`year` - year год изготовления

`kmage` - kmage пробег

Class Car – комментарий к методу

```
27
28      /**
29       * Возвращает текущий пробег автомобиля
30       *
31       * @return это пробег автомобиля
32       */
33      public int getKmage() {
34          return kmage;
35      }
36
37      public String getMake() {
38          return make;
39      }
```

Class Car – комментарий к методу (Javadoc)

Method Summary

Methods

Modifier and Type	Method and Description
void	drive(int km)
int	getKmage() Возвращает текущий пробег автомобиля
java.lang.String	getMake()
int	getYear()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class Car – комментарий к методу (Javadoc)

Method Detail

getKmage

```
public int getKmage()
```

Возвращает текущий пробег автомобиля

Returns:

это пробег автомобиля

getMake

```
public java.lang.String getMake()
```

Class Car – а здесь еще нет комментариев

```
40
41      public int getYear() {
42          return year;
43      }
44
45      public void drive(int km) {
46          if (km > 0)
47              kmage += km;
48      }
49  }
```

Class Car – комментарий к методу (Javadoc)

getYear

```
public int getYear()
```

drive

```
public void drive(int km)
```

Комментарии к полям

- Документировать следует лишь общедоступные поля – обычно они представляют собой статические константы. Например:

```
/**
 * максимальное количество автомобилей
*/
public static final int NumberCar = 30;
```

```
* См. Кей Хорстманн, Гари Корнелл, Java 2. т.1, Основы
```

Комментарии к полям (Javadoc)

Field Detail

NumberCar

```
public static final int NumberCar
```

NumberCar - the maximum number of cars

See Also:

Constant Field Values

Version Tag

- *Form*: @**version** description
- *Used Where*: Interface and Class comments.
- *Used For:* Описание номера текущей версии исходного кода. Часто это просто номер версии, включая только основной и дополнительный номер, а не номер сборки. В некоторых случаях также содержит дату.

- @version 1.32, 08/26/99

Author Tag

```
 3  /**
 4   * Этот класс описывает объекты Поезд (Train).
 5   * Поезд имеет имя и скорость в км/час
 6   * (в пределах от 0 до константы MAX_SPEED).
 7   * Поезда будут использоваться
 8   *
 9   * @author V.V. Sidorik
10   * @version 2.3,  08/06/2016
11   */
12  public class Train {
13      // поля экземпляров
14      String name;
15      int speed;
16      final int MAX_SPEED = 500;
17      // ...
```

Parameter Tag

- *Form*: @**param** name description
- *Used Where*: Method comments.
- *Used For*: Описывает параметр метода. Имя должно быть именем формального параметра. Описание должно быть кратким описанием параметра в одной строке.

- @param obj the object to insert
- @param index the position to insert it at

Param Tag

```
19      /**
20       * Конструируем новый поезд с именем Name
21       * и скоростью равной нулю
22       *
23       * @param Name это имя поезда.
24       */
25      public Train(String Name) {
26          this.name = Name;
27          speed = 0;
28      }
```

Return Tag

- *Form:* **@return** description
- *Used Where:* Method comments.
- *Used For:* Описание возвращаемого значения из метода, за исключением void методов и конструкторов.

- @return <code>true</code> if the insertion is successful, or <code>false</code> if the list already contains the specified object.

Return Tag

```
30      /**
31       * Возвращает текущую скорость поезда.
32       *
33       * @return это текущая скорость.
34       */
35      public int getSpeed() {
36          return speed;
37      }
```

Exception Tag

- *Form:* **@throws exception** description
- *Used Where:* Method comments.
- *Used For:* показывает любые исключения, которые метод может выбросить и возможные причины, по которым эти исключения происходят.

@exception java.io.FileNotFoundException If the specified file could not be found

Exception Tag

```
39      /**
40       * Увеличение скорости поезда
41       *
42       * @param moreMPH - на это значение увеличивается
43       *                  скорость поезда
44       * @throws IllegalArgumentException если новое значение скорости
45       *                                  больше чем MAX_SPEED.
46       */
47      public void accelerate(int moreMPH) {
48          checkMaxSpeed(speed + moreMPH);
49          speed += moreMPH;
50      }
```

That’s all