Утилита javadoc

Комментарий документации начинается с последовательности символов /** и заканчивается последовательностью */. Комментарии документации позволяют добавлять в программу информацию о ней самой с помощью утилиты javadoc (входит в состав JDK). Эта информация помещается в HTML-файл.

Комментарий документации вставляется перед каждым методом, информацию о котором требуется поместить в документацию.

Дескрипторы:

1. **@author** - документирует автора класса.

Синтаксис: **@author** ***описание***

При выполнении javadoc нужно задать опцию **-author**, чтобы поле **@author** включить в HTML-документацию.

2. **{@code}** - позволяет встраивать в комментарий текст (например, фрагмент кода).

Синтаксис: **{@code фрагмент_кода}**

Этот текст будет отображаться с помощью шрифта кода без последующей обработки (например, без HTML-визуализации).

3. **{@inheritDoc}** - наследует комментарий от непосредственного суперкласса.

4. **@exception** - описывает исключение для данного метода.

Синтаксис: **@exception имя_исключения пояснение**

имя_исключения - полное имя исключения

пояснение - строка, которая описывает, в каких случаях может возникнуть данное исключение.
Дескриптор **@exception** используется только для документирования методов.

5. **{@link}** - предлагает встроенную ссылку на дополнительную информацию.

Синтаксис: **{@link пакет.класс#член_класса текст}**
пакет. класс#член_класса - имя класса или метода, на который добавляется ссылка

текст - отображаемая строка

6. **{@literal}** - позволяет встраивать текст в комментарий.
Текст отображается «как есть» без последующей обработки.

Синтаксис: **{@literal описание}**

описание - встраиваемый текст.

7. **@param** - документирует параметр для метода или
параметр типа для класса или интерфейса.

Синтаксис: **@param имя_параметра пояснение**

имя_параметра представляет имя параметра.
Назначение этого параметра определяется соответствующим **пояснением**.

Дескриптор **@param** может использоваться
только для документирования метода, конструктора,
обобщенного класса или интерфейса.

8. **@return** описывает возвращаемое значение метода.
Синтаксис: **@return пояснение**
пояснение - описывает тип и смысл значения, возвращаемого методом.
Дескриптор @return может использоваться только для документирования метода.

9. @see обеспечивает ссылку на дополнительную информацию.
@see привязка
@see пакет.класс#член _класса текст
привязка - ссылка на URL адрес.
пакет. класс#член_класса - имя элемента
текст - представляет отображаемый для данного элемента текст. Текстовый параметр необязателен, имя члена класса тоже является необязательным.

10. **@version** - определяет версию класса.
Синтаксис: **@version информация**
информация представляет строку, содержащую информацию о версии (номер версии). При выполнении утилиты javadoc вам нужно будет
указать опцию **-version**, чтобы поле @version включить в HTML документацию.

Дескрипторы записываются в комментарии документации

```
/* *
*
*   ……..
*
*
*/
```

javadoc -author -version SquareNum.java

```
import java.io.*;
/**
* Этот класс демонстрирует применение комментариев
* документации.
* @author герберт Шилдт (Herbert Schildt)
* @version 1.2
*/
public class SquareNum{
/**
* Этот метод возвращает квадрат числа.
* Это многострочное описание. Вы можете использовать
* столько строк, сколько будет необходимо.
* @param num Значение, которое необходимо возвести в квадрат.
* @return num Значение, возведенное в квадрат.
*/
public double square(double num) {
return num*num;          }
```

```
/**
* Этот метод вводит число, полученное от пользователя.
* @return Введенное значение в виде double.
* @exception В случае ошибки ввода генерируется исключение
* IOException.
* @see IOException
*/

public double getNumber() throws IOException {
  InputStreamReader isr = new InputStreamReader(System.in);
  BufferedReader inData = new BufferedReader(isr);
  String str;
  str = inData.readLine();
  return (new Double(str)).doubleValue();
}
```

```
/**
* Этот метод демонстрирует square().
* @param args Не используется.
* @exception В случае ошибки ввода генерируется исключение
* IOException.
* @see IOException
*/

public static void main(String args[]) throws IOException {
  SquareNum ob = new SquareNum();
  double val;
  System.out.println("Введите значение для возведения в
квадрат:");
  val = ob.getNumber();
  val = ob.square(val);
  System.out.println("Квадрат числа равен" + val) ; }
  }
```

Пакеты

Пакеты - это контейнеры классов, которые используются для сохранения изолированности пространства имен класса.

Пакет позволяет создать классы с одинаковыми именами не беспокоясь о возможных конфликтах с другими одноименными классами .

Пакеты хранятся в иерархической структуре и явно импортируются в определения новых классов.

Для создания пакета необходимо включить команду **package** в качестве первого оператора исходного файла java.

package Имя_Пакета;

Для хранения пакетов система java использует каталоги файловой системы.

Пакеты

Для хранения пакетов система java использует каталоги файловой системы.

package MyPack;

-это означает, что классы, которые объявлены частью пакета MyPack, должны храниться в каталоге с названием MyPack.

Можно создавать иерархию пакетов:

package Пакет1.Пакет2.Пакет3;

package java.awt.image;

Balance.java

```
package MyPack;

public class Balance{
   int bal;

   public Balance(int param){
         bal=param; }

   public void show(){
         System.out.println("Строка");    }     }
```

Account.java

```
package MyPack;

class Account{
   public static void main(String args[]){
         Balance B=new Balance(10);
         B.show();  }  }
```

Пусть мы находимся в каталоге JAVA.

Для компиляции Account.java и Balance.java должны находиться в каталоге MyPack.

Компиляция:

-в каталоге MyPack : javac Account.java
javac Balance.java

-в каталоге JAVA (на уровень выше MyPack) :
javac MyPack /Account.java
javac MyPack / Balance.java

Запуск программы из каталога JAVA, который на уровень выше:
java MyPack / Account

Защита доступа

Классы и пакеты одновременно служат средствами инкапсуляции, хранилищем пространства имен, областью определения переменных и методов.

JAVA определяет четыре категории видимости членов класса:

1. подклассы в одном пакете;
2. классы в одном пакете, не являющиеся подклассами;
3. подклассы в различных пакетах;
4. классы, которые не находятся в одном пакете и не являются подклассами.

Три спецификатора доступа : private, public и protected предоставляют разнообразные способы создания множества уровней доступа.

Если нужно, чтобы элемент был виден за пределами текущего пакета, но только подклассам данного класса, элемент должен быть объявлен как protected.

Доступ к членам класса

Класс A	private	Спецификатор отсутствует	Protected	Public
Класс A	+	+	+	+
Подкласс класса A в этом же пакете	-	+	+	+
Класс этого же пакета, не являющийся подклассом класса A	-	+	+	+
Подкласс класса A из другого пакета	-	-	+	+
Класс из другого пакета, не являющийся подклассом класса A	-	-	-	+

Если у класса нет спецификатора доступа (**public class** …), то класс доступен внутри текущего пакета. Класс может иметь спецификатор **public –** доступен всем.

Если класс объявлен **public**, то он должен быть единственным **public-**классом в текущем файле, имя файла должно совпадать с именем **public** класса.

Классу, который импортирует пакет и не является подклассом классов пакета, будут доступны только public классы и public члены классов пакета.

Пусть мы находимся в каталоге JAVA (Test.java и каталог MyPack)

Balance.java в каталоге MyPack

```
package MyPack;

public class Balance{
   int bal;

   public Balance(int param){
         bal=param; }

  public void show(){
     System.out.println(“Строка”);
 } }
```

Test.java

```
import MyPack.*;

class Test{
   public static void main(String args[]){
         Balance B=new Balance(10);
         B.show();  }  }
```

Создание и использование jar-архивов

Использование jar-архивов предоставляет разработчикам java программ ряд преимуществ:

•**Повышение эффективности загрузки** - вместо нескольких файлов отдельных классов загружается единственный файл JAR-архива.

•**Улучшенное хранилище файлов -** к файлы классов хранятся в одном сжатом файле архива.

•**Повышение защищённости** – в jar-файл можно поместить цифровую подпись, гарантирующую, что файл архива не изменился с момента её внесения.

•**Независимость от платформы -** построение jar-архивов базируется на использовании программы pkzip. jar-файлы могут создаваться и сохраняться на любой компьютерной платформе.

Создание JAR-архива

JAR-файл отличается от zip-файла наличием дополнительного текстового файла описания (manifest file). Этот файл содержит сведения обо всех помещённых в данный архив файлах.

В состав файла описания входит:

•Номер версии стандарта JAR (Manifest-Version)

•Минимальный номер версии утилиты JAR, которая сможет прочитать этот архив. (этот параметр необязателен и имеет название Required-Version).

•Отдельная запись для любого, помещённого в архив файла. Необязательно перечислять все помещённые в архив файлы, достаточно указать только файлы главных классов.

Утилита jar

Некоторые ключи:

–c - создать новый архив.

–t - вывести содержание указанного архивного.

–x - извлечь файлы, указанные в списке, или извлечь все файлы.

–f - указывает, что имя архивного файла помещено первым в списке.

–v - указывает, что утилита должна сопровождать сообщениями выполнение всех действий.

–0 -сохранение файлов в архиве выполняется без их сжатия.

–u - указывает, что нужно обновить указанные файлы.

–e - точка входа (файл для запуска программы, содержит main()).

jar cvfe TTT.jar Test Test.class MyPack/Balance.class

jar cvfe TTT.jar Test Test.* MyPack/*.class MyPack/*.java

Запуск на выполнение:

java –jar TTT.jar

```
import java.io.*;
class Read {
  public static void main(String args[]) throws IOException {
       char c;
      String str;
      double A;
      int B;
// создаем буфер на входном потоке
 BufferedReader BR= new BufferedReader(new InputStreamReader(System.in));

      System.out.println("Вводите символы, ‘q‘ - для выхода.");

      do {
            c = (char)br.read();        // считываем символ из буфера
            str = BR.readLine();       // считываем строку
            A = Double.parseDouble(str);  // преобразуем строку в число
            str = BR.readLine();           // снова считываем строку
            B = Integer.parseInt(str);  // преобразуем  строку в число
           System.out.println(c+ “   “ + A+”  “ + B );
   } while(c != 'q') ;
 }  }
```

```
import java.util.*;

class AvgNums {
   public static void main(String args[ ]) {
      Scanner in = new Scanner( System.in );  // объект форматированного ввода
      int count = 0;
      double sum = 0.0;
      System.out.println("Введите числа:");

      while(in.hasNext()) {  // проверка на наличие символов
         if( in.hasNextDouble() ){   // если это double
            sum += in.nextDouble( );  count++;    }  // получаем и прибавляем

        else {  String str = in.next( ); // получаем следующее значение
                 if( str.equals("выход") )  break;
                else {
                     System.out.println ("Ошибка формата данных") ;
                     return;  }  }
        }
        System. out. println ("Среднее равно" + sum / count);
} }
```