Класс FileReader

Этот класс является производным от класса Reader и служит для чтения содержимого файла. Ниже приведены два наиболее употребительных конструктора этого класса.

```
FileReader(String путь_к_файлу)
FileReader(File объект_файла)
```

Здесь параметр *путь_к_файлу* обозначает имя полного пути к файлу, а параметр *объект_файла* — объект типа File, описывающий файл. Оба конструктора могут сгенерировать исключение типа FileNotFoundException.

В приведенном ниже примере программы показано, как организовать построчное чтение и запись данных из файла в стандартный поток вывода. Программа читает собственный исходный файл, который должен находиться в текущем каталоге.

```
// Продемонстрировать применение класса FileReader
// В этой программе используется оператор try с ресурсами.
// Требуется установка комплекта JDK, начиная с версии 7

import java.io.*;

class FileReaderDemo {
    public static void main(String args[]) {
        try ( FileReader fr = new FileReader("FileReaderDemo.java") )
        {
            int c;

            // прочитать и вывести содержимое файла
            while((c = fr.read()) != -1) System.out.print((char) c);

        } catch(IOException e) {
            System.out.println("Ошибка ввода-вывода: " + e);
        }
    }
}
```

Класс FileWriter

Этот класс создает поток вывода типа Writer для записи данных в файл. Ниже приведены наиболее употребительные конструкторы класса FileWriter.

```
FileWriter(String путь_к_файлу)
FileWriter(String путь_к_файлу, boolean добавить)
FileWriter(File объект_файла)
FileWriter(File объект_файла, boolean присоединить)
```

Здесь параметр *путь_к_файлу* обозначает имя полного пути к файлу, а параметр *объект_файла* — объект типа File, описывающий файл. Если параметр *присоединить* принимает логическое значение true, то выводимые данные присоединяются в конце файла. Все конструкторы данного класса могут генерировать исключение типа IOException.

Создание объекта типа FileWriter не зависит от того, существует ли файл. Когда создается объект типа FileWriter, то попутно создается и файл, прежде

чем открыть его для вывода. Если же предпринимается попытка открыть файла, доступный только для чтения, то генерируется исключение типа IOException.

В приведенном ниже примере представлена переделанная под ввод-вывод символов версия программы из рассмотренного ранее примера, демонстрировавшего применение класса FileOutputStream. В этой версии организуется буфер символов для хранения образца текста. С этой целью сначала создается объект типа String, а затем вызывается метод getChars() для извлечения эквивалентного символьного массива. Далее создаются три файла. Первый файл, file.txt, должен содержать каждый второй символ из образца текста, второй файл, file2.txt, — все символы из образца текста, а третий файл, file3.txt, — только последнюю четверть символов из образца текста.

```
// Продемонстрировать применение класса FileWriter
// В этой программе используется оператор try с ресурсами.
// Требуется установка комплекта JDK, начиная с версии 7

import java.io.*;

class FileWriterDemo {
    public static void main(String args[]) throws IOException {
        String source = "Now is the time for all good men\n" +
                        " to come to the aid of their country\n" +
                        " and pay their due taxes.";
        char buffer[] = new char[source.length()];
        source.getChars(0, source.length(), buffer, 0);

        try ( FileWriter f0 = new FileWriter("file1.txt");
              FileWriter f1 = new FileWriter("file2.txt");
              FileWriter f2 = new FileWriter("file3.txt") )
        {
            // вывести символы в первый файл
            for (int i=0; i < buffer.length; i += 2) {
                f0.write(buffer[i]);
            }

            // вывести символы во второй файл
            f1.write(buffer);

            // вывести символы в третий файл
          f2.write(buffer,buffer.length-buffer.length/4,buffer.length/4);
        } catch(IOException e) {
            System.out.println("Произошла ошибка ввода-вывода");
        }
    }
}
```

Класс CharArrayReader

Класс CharArrayReader реализует поток вывода, использующий массив в качестве источника данных. У этого класса имеются два конструктора, каждый из которых принимает массив символов в качестве источника данных.

```
CharArrayReader(char массив[])
CharArrayReader(char массив[], int начало, int количество_символов)
```

Здесь параметр *массив* обозначает источник ввода данных. Второй конструктор создает объект класса, производного от класса `Reader`, из подмножества массива символов, начинающегося с позиции, обозначаемой параметром *начало*, и длиной, определяемой параметром *количество_символов*.

Метод `close()`, реализуемый классом `CharArrayReader`, не генерирует исключений. Это связано с тем, что его вызов не может завершиться неудачно. В следующем примере применяется пара объектов класса `CharArrayReaders`:

```
// Продемонстрировать применение класса CharArrayReader
// В этой программе используется оператор try с ресурсами.
// Требуется установка комплекта JDK, начиная с версии 7

import java.io.*;

public class CharArrayReaderDemo {
    public static void main(String args[]) {
        String tmp = "abcdefghijklmnopqrstuvwxyz";
        int length = tmp.length();
        char c[] = new char[length];

        tmp.getChars(0, length, c, 0);
        int i;

        try (CharArrayReader input1 = new CharArrayReader(c))
        {
            System.out.println("input1:");
            while((i = input1.read()) != -1) {
                System.out.print((char)i);
            }
            System.out.println();
        } catch(IOException e) {
            System.out.println("Ошибка ввода-вывода: " + e);
        }

        try (CharArrayReader input2 = new CharArrayReader(c, 0, 5))
        {
            System.out.println("input2:");
            while((i = input2.read()) != -1) {
                System.out.print((char)i);
            }
            System.out.println();
        } catch(IOException e) {
            System.out.println("Ошибка ввода-вывода: " + e);
        }
    }
}
```

Объект `input1` создается с использованием всего английского алфавита в нижнем регистре, в то время как объект `input2` содержит только первые пять букв. Эта программа выводит следующий результат:

```
input1:
abcdefghijklmnopqrstuvwxyz
input2:
abcde
```

Класс CharArrayWriter

Класс CharArrayWriter реализует поток вывода, использующий массив в качестве адресата для выводимых данных. У класса CharArrayWriter имеются два конструктора:

```
CharArrayWriter()
CharArrayWriter(int количество_символов)
```

В первой форме конструктора создается буфер размером, выбираемым по умолчанию. Во второй форме буфер создается размером, задаваемым параметром *количество_символов*. Буфер находится в поле buf класса CharArrayWriter. Размер буфера будет последовательно увеличиваться по мере надобности. Количество байтов, содержащихся в буфере, находится в поле count того же класса. Оба поля, buf и count, являются защищенными.

Метод close() не оказывает никакого влияния на класс CharArrayWriter. В приведенном ниже примере представлена переделанная под ввод-вывод символов версия программы из рассмотренного ранее примера, демонстрировавшего применение класса ByteArrayOutputStream. А в этой версии демонстрируется применение класса CharArrayWriter, хотя выводимый результат оказывается таким же, как и в предыдущей версия.

```
// Продемонстрировать применение класса CharArrayWriter
// В этой программе используется оператор try с ресурсами.
// Требуется установка комплекта JDK, начиная с версии 7

import java.io.*;
class CharArrayWriterDemo {
    public static void main(String args[]) throws IOException {
        CharArrayWriter f = new CharArrayWriter();
        String s = " Эти данные должны быть выведены в массив";
        char buf[] = new char[s.length()];

        s.getChars(0, s.length(), buf, 0);

        try {
            f.write(buf);
        } catch(IOException e) {
            System.out.println("Ошибка записи в буфер");
            return;
        }

        System.out.println("Буфер в виде символьной строки");
        System.out.println(f.toString());
        System.out.println("В массив");

        char c[] = f.toCharArray();
        for (int i=0; i<c.length; i++) {
            System.out.print(c[i]);
        }

        System.out.println("\nВ поток вывода типа FileWriter()");

        // использовать оператор try с ресурсами для управления
        // потоком ввода-вывода в файл
        try ( FileWriter f2 = new FileWriter("test.txt") )
        {
```

```
            f.writeTo(f2);
        } catch(IOException e) {
            System.out.println("Ошибка ввода-вывода: " + e);
        }

        System.out.println("Установка в исходное состояние");
        f.reset();

        for (int i=0; i<3; i++) f.write('X');

        System.out.println(f.toString());
    }
}
```

Класс BufferedReader

Класс BufferedReader увеличивает производительность благодаря буферизации ввода. У него имеются следующие два конструктора:

```
BufferedReader(Reader поток_ввода)
BufferedReader(Reader поток_ввода, int размер_буфера)
```

В первой форме конструктора создается буферизованный поток ввода символов, использующий размер буфера по умолчанию. Во второй форме конструктора задается *размер_буфера*.

Закрытие потока типа BufferedReader приводит также к закрытию базового потока, определяемого параметром *поток_ввода*. Аналогично потоку ввода байтов, буферизованный поток ввода символов также поддерживает механизм перемещения обратно по потоку ввода в пределах доступного буфера. Для этой цели в классе BufferedReader реализуются методы mark() и reset(), а метод BufferedReader.markSupported() возвращает логическое значение true. В версии JDK 8 класс BufferedReader дополнен новым методом lines(). Этот метод возвращает ссылку типа Stream на последовательность строк, введенных из потока чтения. (Класс Stream входит в состав прикладного программного интерфейса API потоков данных, обсуждаемого в главе 29.)

В приведенном ниже примере представлена переделанная под ввод-вывод символов версия программы из рассмотренного ранее примера, демонстрировавшего применение класса BufferedInputStream. В новой версии демонстрируется применение класса BufferedReader для организации потока буферизированного ввода. Как и прежде, для синтаксического анализа с целью обнаружить ссылку на элемент HTML-разметки знака авторского права в данной версии программы используются методы mark() и reset(). Такая ссылка начинается со знака амперсанда (&) и оканчивается точкой с запятой (;) без всяких промежуточных пробелов. Образец введенных данных содержит два амперсанда, чтобы наглядно показать, когда происходит установка в исходное состояние с помощью метода reset() и когда этого не происходит. Эта версия программы выводит такой же результат, как и предыдущая ее версия:

```
// Использовать буферизованный ввод.
// В этой программе применяется оператор try с ресурсами.
// Требуется установка комплекта JDK, начиная с версии 7
```

```
import java.io.*;

class BufferedReaderDemo {
    public static void main(String args[]) throws IOException {
        String s = " Это знак авторского права &copy; " +
                   ", а &copy – нет.\n";
        char buf[] = new char[s.length()];
        s.getChars(0, s.length(), buf, 0);

        CharArrayReader in = new CharArrayReader(buf);
        int c;
        boolean marked = false;

        try ( BufferedReader f = new BufferedReader(in) )
        {

            while ((c = f.read()) != -1) {
                switch(c) {
                    case '&':
                        if (!marked) {
                            f.mark(32);
                            marked = true;
                        } else {
                            marked = false;
                        }
                        break;
                    case ';':
                        if (marked) {
                            marked = false;
                            System.out.print("(c)");
                        } else
                            System.out.print((char) c);
                        break;
                    case ' ':
                        if (marked) {
                            marked = false;
                            f.reset();
                            System.out.print("&");
                        } else
                            System.out.print((char) c);
                        break;
                    default:
                        if (!marked)
                        System.out.print((char) c);
                        break;
                }
            }
        } catch(IOException e) {
            System.out.println("Ошибка ввода-вывода: " + e);
        }
    }
}
```

Класс BufferedWriter

Класс `BufferedWriter` является производным от класса `Writer` и буферизует выводимые данные. Применяя класс `BufferedWriter`, можно повысить производительность за счет снижения количества операций физической записи в устройство вывода.

У класса BufferedWriter имеются два конструктора:

```
BufferedWriter(Writer поток_вывода)
BufferedWriter(Writer поток_вывода, int размер_буфера)
```

В первой форме конструктора создается буферизованный поток вывода, использующий буфер размером, выбираемым по умолчанию. А во второй форме задается конкретный *размер_буфера*.

Класс PushbackReader

Класс PushbackReader позволяет возвратить в поток ввода один или больше символов, чтобы просматривать этот поток, не вводя из него данные. Ниже приведены два конструктора данного класса.

```
PushbackReader(Reader поток_ввода)
PushbackReader(Reader поток_ввода, int размер_буфера)
```

В первой форме конструктора создается буферизованный поток ввода, в который можно возвратить один символ, а во второй задается конкретный *размер_буфера* для возврата символов обратно в поток ввода.

При закрытии потока типа PushbackReader закрывается также базовый поток, определяемый параметром *поток_ввода*. В классе PushbackReader предоставляется метод unread(), возвращающий один или больше символов в вызывающий поток ввода. Ниже приведены три общие формы объявления этого метода.

```
void unread(int символ) throws IOException
void unread(char буфер[]) throws IOException
void unread(char буфер[], int смещение, int количество_символов)
     throws IOException
```

В первой форме в поток ввода возвращается указанный *символ*. Это будет следующий символ, возвращаемый при последующем вызове метода read(). Во второй форме в поток ввода возвращаются символы из указанного *буфера*. А в третьей форме в поток ввода возвращается заданное *количество_символов* из указанного *буфера*, начиная с позиции *смещение*. Исключение типа IOException генерируется при попытке возвратить символ в поток ввода, когда буфер возврата заполнен.

В приведенном ниже примере представлена переделанная версия программы из рассмотренного ранее примера, демонстрировавшего применение класса PushbackInputStream. В новой версии демонстрируется применение класса PushbackReader, но, как и прежде, данный пример показывает, как возврат данных (в данном случае символов) в поток ввода можно использовать в синтаксическом анализаторе языка программирования для различения операций сравнения (==) и присваивания (=). Результат выполнения данной версии программы такой же, как и в прежней ее версии.

```
// Продемонстрировать применение метода unread()
// из класса PushbackInputStream.
// В этой программе применяется оператор try с ресурсами.
// Требуется установка комплекта JDK, начиная с версии 7
import java.io.*;

class PushbackReaderDemo {
```

```
    public static void main(String args[]) {
        String s = "if (a == 4) a = 0;\n";
        char buf[] = new char[s.length()];
        s.getChars(0, s.length(), buf, 0);
        CharArrayReader in = new CharArrayReader(buf);

        int c;

        try ( PushbackReader f = new PushbackReader(in) )
        {
            while ((c = f.read()) != -1) {
                switch(c) {
                    case '=':
                        if ((c = f.read()) == '=')
                            System.out.print(".eq.");
                        else {
                            System.out.print("<-");
                            f.unread(c);
                        }
                        break;
                    default:
                        System.out.print((char) c);
                        break;
                }
            } catch(IOException e) {
                System.out.println("Ошибка ввода-вывода: " + e);
            }
        }
    }
}
```

Класс PrintWriter

Класс PrintWriter, по существу, является символьной версией класса PrintStream. Он реализует интерфейсы Appendable, Closeable и Flushable. У класса PrintWriter имеется несколько конструкторов. Рассмотрим сначала следующие формы конструкторов этого класса:

```
PrintWriter(OutputStream поток_вывода)
PrintWriter(OutputStream поток_вывода, boolean автоочистка)
PrintWriter(Writer поток_вывода)
PrintWriter(Writer поток_вывода, boolean автоочистка)
```

где параметр *поток_вывода* обозначает открытый поток вывода типа OutputStream, который будет принимать выводимые данные. Параметр *автоочистка* определяет, будет ли буфер вывода автоматически очищаться всякий раз, когда вызывается метод println(), printf() или format(). Если параметр *автоочистка* принимает логическое значение true, то происходит автоматическая очистка буфера вывода. А если этот параметр принимает логическое значение false, то очистка буфера вывода не производится автоматически. Конструкторы, не принимающие параметр *автоочистка*, не производят очистку буфера вывода автоматически.

Следующий ряд конструкторов предоставляет простую возможность создать объект класса PrintWriter для вывода данных в файл:

```
PrintWriter(File файл_вывода) throws FileNotFoundException
PrintWriter(File файл_вывода, String набор_символов)
    throws FileNotFoundException, UnsupportedEncodingException
PrintWriter(String имя_файла_вывода) throws FileNotFoundException
PrintWriter(String имя_файла_вывода, String набор_символов)
    throws FileNotFoundException, UnsupportedEncodingException
```

Эти конструкторы позволяют создать объект класса `PrintWriter` из объекта типа `File` или по имени файла. Но в любом случае файл создается автоматически. Любой существующий файл с тем же именем уничтожается. Как только поток вывода будет создан в виде объекта класса `PrintWriter`, он будет направлять все выводимые данные в указанный файл. Конкретную кодировку символов можно задать в качестве параметра *набор_символов*.

Класс `PrintWriter` предоставляет методы `print()` и `println()` для всех типов, включая тип `Object`. Если аргумент не относится к примитивному типу, методы из класса `PrintWriter` вызывают сначала метод `toString()` такого объекта, а затем выводят результат его выполнения.

В классе `PrintWriter` поддерживается также метод `printf()`. Он действует точно так же, как и в описанном ранее классе `PrintStream`, позволяя задать точный формат данных. Метод `printf()` объявляется в классе `PrintWriter` следующим образом:

```
PrintWriter printf(String форматирующая_строка, Object ... аргументы)
PrintWriter printf(Locale региональные_настройки,
            String форматирующая_строка, Object ... аргументы)
```

В первой форме данного метода заданные *аргументы* выводятся в стандартный поток вывода в формате, указанном в качестве параметра *форматирующая_строка*, с учетом региональных настроек по умолчанию. А во второй форме можно указать конкретные региональные настройки. Но в любом случае возвращается вызывающий поток вывода в виде объекта типа `PrintWriter`.

В классе `PrintWriter` поддерживается также метод `format()`. Ниже приведены общие формы данного метода. Этот метод действует подобно методу `printf()`.

```
PrintWriter format(String форматирующая_строка, Object ... аргументы)
PrintWriter format(Locale региональные_настройки,
                   String форматирующая_строка, Object ... аргументы)
```

Класс `Console`

Класс `Console` был введен в состав пакета `java.io` в версии JDK 6. Он служит для ввода-вывода данных на консоль, если таковая имеется, и реализует интерфейс `Flushable`. Класс `Console` является служебным, поскольку он функционирует главным образом через стандартные потоки ввода-вывода `System.in` и `System.out`. Тем не менее он упрощает некоторые виды консольных операций, особенно при чтении символьных строк с консоли.

Конструкторы в классе `Console` не предоставляются. Его объект получается в результате вызова метода `System.console()`, как показано ниже.

```
static System.console()
```