Введення-виведення даних в Java

Информация на устройствах внешней памяти хранится в **файлах** – именованных областях на диске или другом носителе. Структура размещения файлов на носителях информации, а также виды и правила задания атрибутов файла (имени, типа, даты создания и/или модификации и т.п.) называется **файловой системой**.

Файловые системы в разных операционных системах, как правило, отличаются друг от друга. Так, при выполнении Java-программы в среде Unix элементы пути должны отделяться прямой косой чертой "/". В компьютерах под управлением Windows элементы пути разделяются символами обратной косой черты "\". Для этой же цели на компьютерах Macintosh используется символ двоеточия "**:**". Кроме того, могут существовать различные ограничения на длину имен файлов и каталогов.

Абстрагироваться от всех зависимых от платформы элементов файлового ввода-вывода можно с помощью класса **File**.

Класс **File** позволяет получить от системы все данные, начиная с имени файла и заканчивая временем последней его модификации. Его можно использовать для создания новых каталогов, а также для удаления и переименования файлов. Для создания объекта **File** нужно вызвать один из трех конструкторов класса:

File(String path)	создает объект **File** с указанным полным именем файла
File(String path, String name)	создает объект **File**, используя отдельно путь и имя файла
File(File dir, String name)	создает объект **File**, используя путь и имя файла, при этом путь определяется другим объектом **File**.

Методы класса **File**:

public String getName()	определяет имя файла
public String getPath()	возвращает относительный путь к файлу
public String getAbsolutePath()	возвращает полное имя файла
public String getParent()	возвращает родительский каталог файла
public boolean exists()	проверяет, существует ли данный файл
public boolean isFile()	проверяют, является ли он файлом или каталогом
public boolean isDirectory()	
public boolean isAbsolute()	проверяет, задано ли его имя как абсолютное имя

public boolean canRead() **public boolean canWrite()**	проверяют можно ли читать данный файл и можно ли записывать в него данные;
public long length() **public long lastModified()**	возвращают характеристики файла: длину и время последней модификации
public boolean mkdir()	создание каталога
public boolean mkdirs()	создание дерева каталогов
public boolean renameTo(File newName)	переименование файла или каталога
public boolean delete()	удаление файла или каталога
public String[] list()	создает массив имен файлов и подкаталогов в каталоге в виде строк
public File[] listFiles()	создает массив имен файлов и подкаталогов в каталоге в виде объектов класса **File**.

Організація введення-виведення в Java

Потоком называется канал обмена информацией между ее источником и получателем.

На одном конце потока всегда находится программа на Java. Если она будет служить источником данных, то данный поток будет выходным, если программа будет находиться на принимающей стороне – то входным.

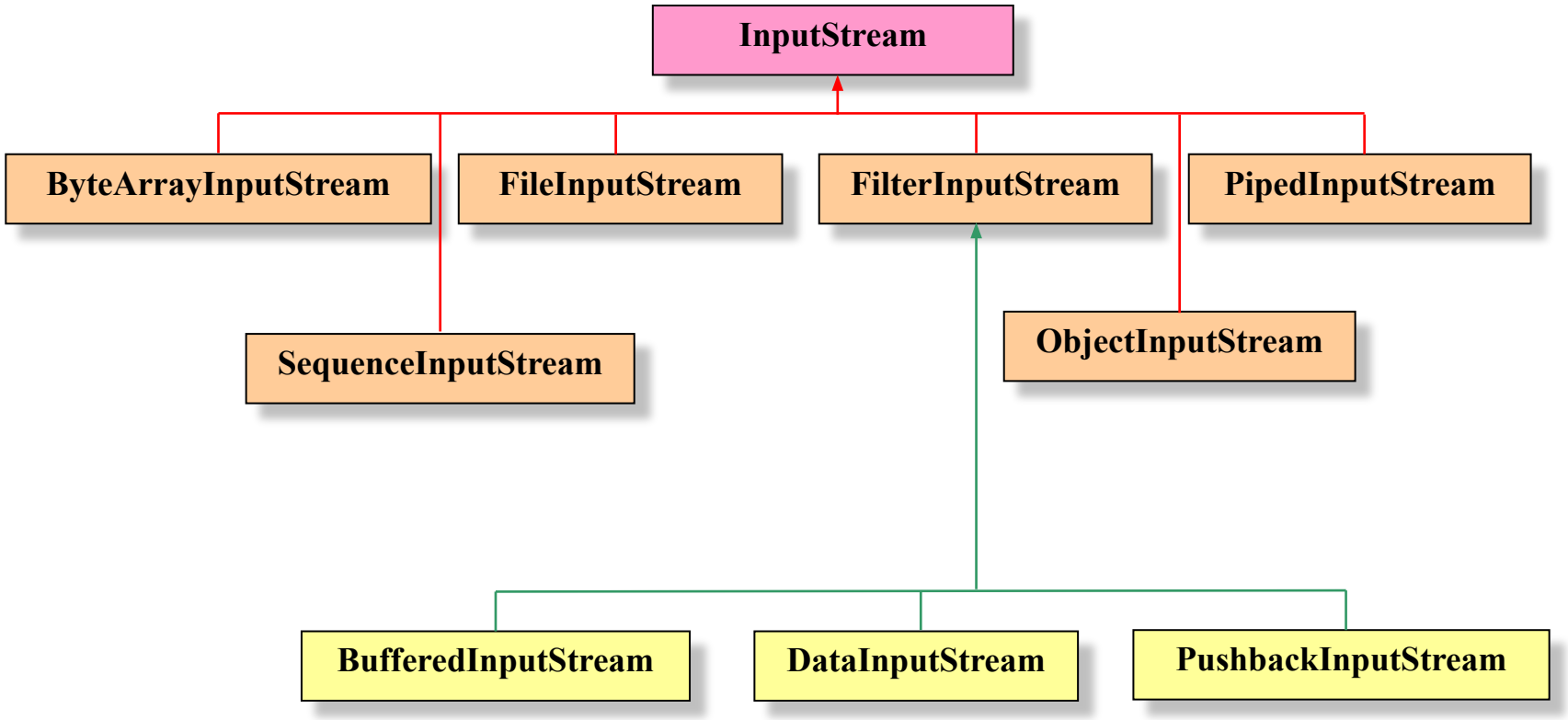
В Java определены два типа потока: байтовый и символьный потоки.

Байтовый поток оперирует с байтами и используется при чтении (***input stream***) или записи (***output stream***) данных в двоичном коде.

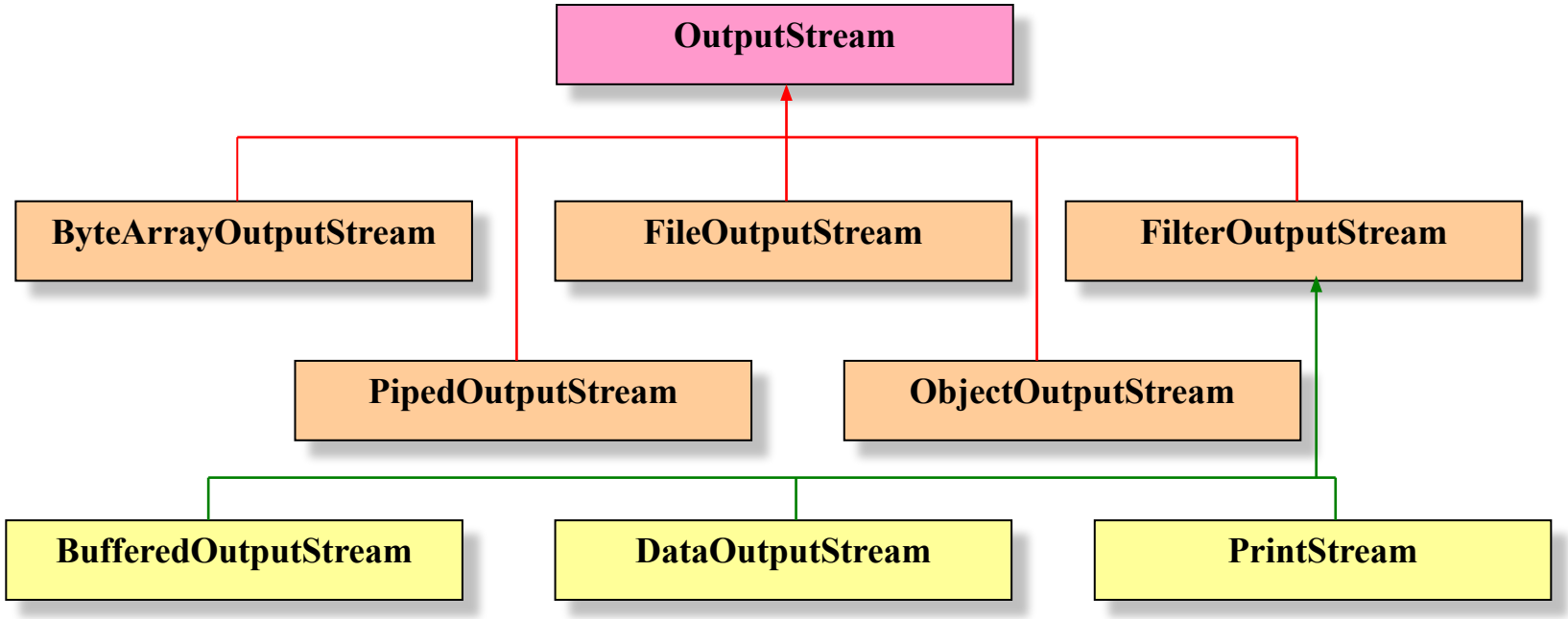
Символьный поток используется для ввода и вывода символов. В Java символ не является эквивалентом байта, так как представляет собой 16-битовый элемент, предназначенный для размещения кодов Unicode.

В языке Java потоки представляются **классами**. От базовых классов порождаются другие классы, в большей степени ориентированные на конкретный тип ввода или вывода.
Все классы ввода-вывода Java описаны в пакете **java.io**.

Ієрархії класів для вхідних байтових потоків



Ієрархії класів для вихідних байтових потоків



Класи InputStream і OutputStream

Абстрактный класс **InputStream** представляет собой базовый поток ввода.

Основным методом класса **InputStream** является **read()**. Этот метод осуществляет передачу байтов данных из входного потока в массив, размещенный в памяти.

Абстрактный класс **OutputStream**, обеспечивает базовые функции для всех выходных потоков

Основным методом класса **OutputStream** является **write()**

Класи FileInputStream і FileOutputStream

Класс **FileInputStream** создает (открывает) объект, который можно использовать для чтения байтов из файла

Класс **FileOutputStream** создает объект **OutputStream**, который можно применять для записи байтов в файл.

Класи ByteArrayInputStream і ByteArrayOutputStream

Класс **ByteArrayInputStream** является реализацией входного потока, которая использует байтовый массив как источник.

При выполнении операции вывода массив байтов можно переслать в локальную память компьютера. Конкретным классом выводного потока **OutputStream**, способным решить эту задачу, является **ByteArrayOutputStream**.

Клас SequenceInputStream

Класс **SequenceInputStream** позволяет сцеплять множество объектов входного потока. Основная форма конструктора этого класса использует в качестве параметров два объекта класса **InputStream** или его подклассов.

Класс выполняет запросы чтения первого объекта типа **InputStream**, пока он не закончится, и затем переключается на второй. В свою очередь, используя в качестве одного из объектов объект класса **SequenceInputStream**, можно организовать ввод более чем двух объектов входного потока.

Класи FilterInputStream і FilterOutputStream

Фильтрованные потоки — просто оболочки (wrappers) вокруг основных потоков ввода или вывода, которые прозрачно обеспечивают некоторый расширенный уровень функциональных возможностей. Доступ к этим потокам обычно выполняется с помощью методов, использующих порождающий поток, связанный с суперклассом фильтрованных потоков. Типичные применения фильтрованных потоков — буферизация, трансляция символов и необработанных данных. Фильтрованные байтовые входные и выходные потоки реализованы соответственно в классах **FilterInputStream** и **FilterOutputStream**.

Хотя классы **FilterInputStream** и **FilterOutputStream** реализованы как конкретный (не абстрактные классы), однако они не добавляет никаких функциональных возможностей к тем потокам, на которых он строится. Нет необходимости создавать экземпляры объектов этих классов – достаточно просто использовать их подклассы.

Класи BufferedInputStream і BufferedOutputStream

Байтовый **буферизированный поток** расширяет классы фильтрованного потока, присоединяя буфер памяти к потокам ввода/вывода. Такой буфер позволяет выполнять операции ввода-вывода (используя внутренний буфер) не с одним, а с несколькими байтами одновременно, и, следовательно, увеличивает эффективность работы программы. При наличии буфера становится возможным такие операции, как пропуск байтов, маркировка и переустановка потока. Буферизированные поточные классы – это классы **BufferedInputStream** и **BufferedOutputStream**.

Класи DataInputStream і DataOutputStream

Классы **DataInputStream** и **DataOutputStream** позволяют приложению соответственно читать примитивные типы данных Java из нижележащего входного потока или записывать примитивные типы данных Java в нижележащий выходной поток в виде, не зависящем от компьютерной платформы или операционной системы

Класи PushbackInputStream

Одно из применений буферизации – выполнение возврата считанного байта обратно во входной поток (операция pushback). Эту возможность в Java осуществляет класс **PushbackInputStream**.

Класи PipedInputStream і PipedOutputStream

Вычислительные потоки отличаются от процессов тем, что для них уровень защиты операционной системы от влияния других потоков существенно ниже. При проектировании многопоточных приложений рекомендуется каждый из потоков изолировать в собственном классе, после чего определить каналы, по которым эти процессы будут взаимодействовать между собой. Для того, чтобы позволить одному потоку записать данные в другой поток или считать данные из другого потока, используются классы **PipedInputStream** и **PipedOutputStream**.

Серіалізация об'єктів і використання класівObjectOutputStream і ObjectInputStream

Сериализация (serialization) – это процесс записи состояния объекта в байтовый поток. Этот процесс используется, когда необходимо сохранить состояние программы в постоянной памяти, например, в файле. Сохраненное состояние можно восстановить, используя процесс **десериализации.** Сериализация используется также для реализации механизма вызова удаленных методов (RMI, Remote Method Invocation). Механизм RMI позволяет объекту Java на одной машине вызывать метод другого объекта Java, находящегося на удаленной машине. Удаленный объект можно указывать как аргумент удаленного метода. Посылающая машина сериализирует объект и передает его. Принимающая машина десериализует принятый объект.

Класс **ObjectOutputStream** позволяет записывать данные примитивных типов и образы объектов в выводной поток (сериализовать их).

Класс **ObjectInputStream** читает объекты, предварительно записанные с помощью методов класса **ObjectOutputStream**, из входного потока.

Клас PrintStream

Класс **PrintStream** позволяет вывести в поток нижнего уровня представление данных примитивного, строкового или объектного типа, которое они будут иметь при печати.

Методи класів для вхідних байтових потоків

skip() — пропускает заданное количество байт во входном потоке

available() — возвращает количество байтов, имеющихся в данный момент в потоке

mark() — запоминает положение текущего указателя данных

reset() — изменяет положение точки ввода таким образом, что следующий вызов метода **read()** начнет ввод данных с запомненной ранее позиции

close() — закрывает вводный поток.

Большинство подклассов класса **InputStream** не поддерживает методы **mark()** и **reset()**. Поэтому, прежде чем вызывать эти методы, следует выполнить вызов метода **markSupported()** и определить, поддерживает ли эти методы используемый поток.

Методи класів для вихідних байтових потоків

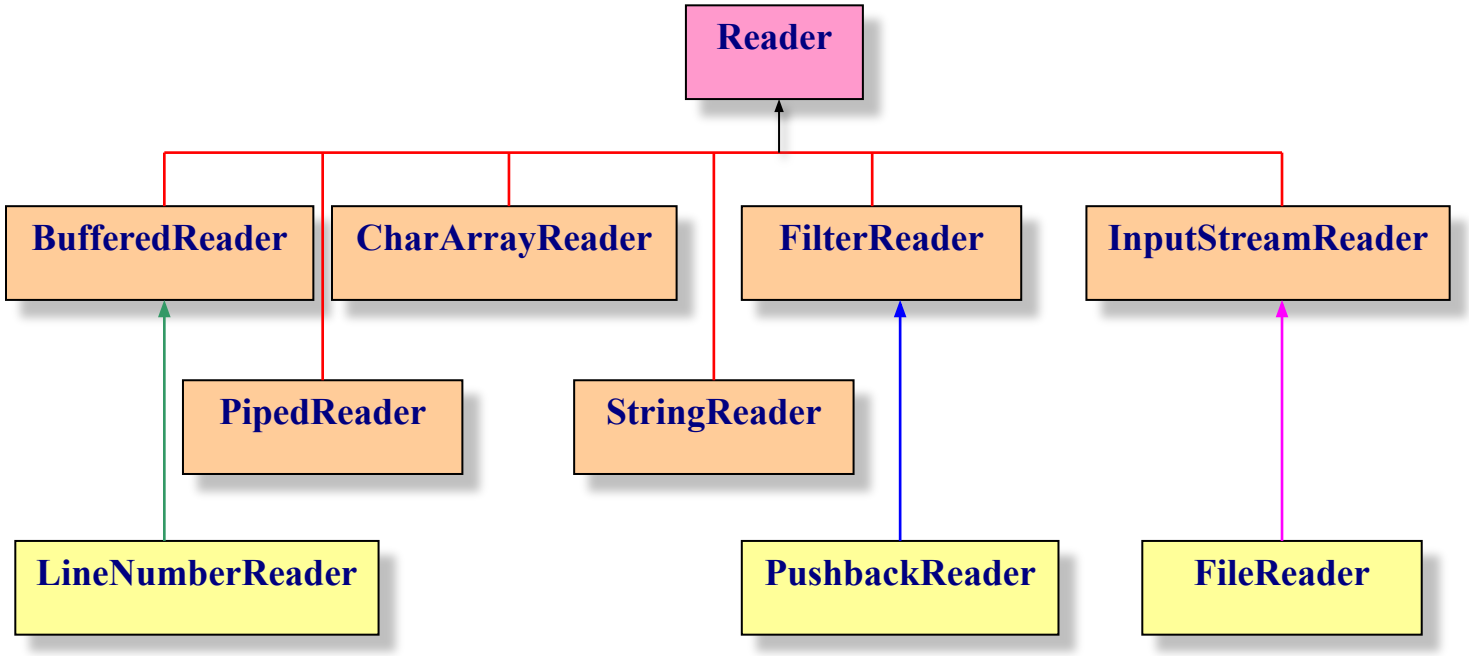
flush() — выполняет принудительную выгрузку помещенных в буфер данных на внешнее устройство

close() — полностью освобождает все ресурсы, использовавшиеся при работе потока

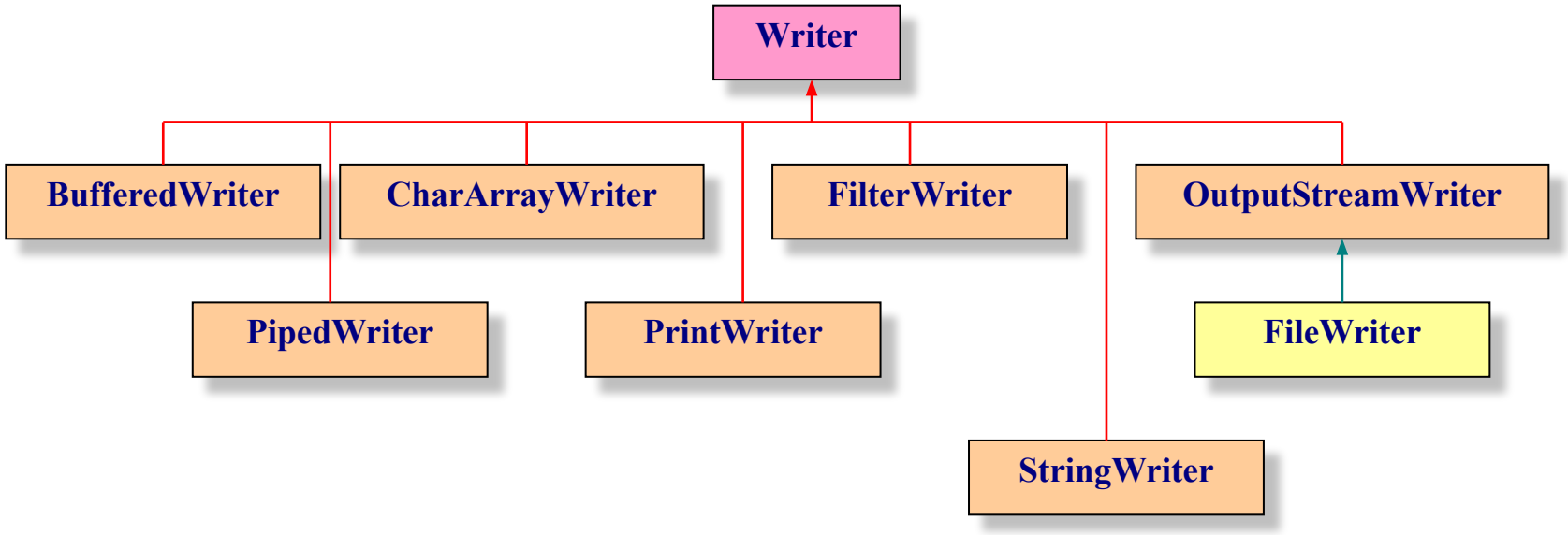
Символьні потоки введення-виведення

Помимо семейств классов побайтового обмена, производных от классов **InputStream** и **OutputStream**, Java предоставляет символьные потоки, предназначенные для обработки символов в кодировке Unicode.

Ієрархія класів символьного введення



Ієрархія класів символьного виводу



Символьные потоки лучше всего использовать для реализации многоязычной поддержки.

Класи Reader і Writer

Функции классов **InputStream** и **OutputStream** для символьных потоков выполняют абстрактные классы **Reader** и **Writer**.

Класс **Reader** определяет модель символьного входного потока.

Методы **skip()**, **mark()**, **reset()**, **markSupported()** и **close()** действуют аналогично таким же методам класса **InputStream**, а вместо метода **available()** используется метод

ready() возвращает значение **true**, если гарантируется, что следующая операция **read()** не будет переведена в состояние ожидания.

Класс **Writer** определяет модель поточного символьного вывода.

Методы **flush()** и **close()** класса **Writer** также действуют аналогично соответствующим методам класса **OutputStream**.

Класи InputStreamReader і OutputStreamWriter

Классы **InputStreamReader** и **OutputStreamWriter** являются переходниками между байтовыми и символьными потоками.

Класс **InputStreamReader** преобразует байтовый поток в символьный поток.

Класс **OutputStreamWriter** преобразует символьный поток в байтовый поток.

Класи FileReader і FileWriter

Класс **FileReader** создает объект, который можно использовать для чтения содержимого файла.

Класс **FileWriter** создает объект, который можно использовать для записи в файл.

Класи CharArrayReader і CharArrayWriter

Класс **CharArrayReader** является реализацией входного потока, которая использует символьный массив как источник.

Класс **CharArrayWriter** реализует выходной поток, записывающий данные в символьный массив.

Класи BufferedReader, LineInputReader і BufferedWriter

Класс **BufferedReader** читает текст из символьного входного потока, используя буферизацию символов, так, чтобы обеспечить эффективное чтение символов, массивов и строк.

Класс **LineNumberReader**, являющийся подклассом **BufferedReader**, наряду с буферизованным вводом, обеспечивает слежение за номерами строк.

Класс **BufferedWriter** выполняет буферизованную запись символов, массивов и строк в выходной поток.

Класи FilterReader, PushbackReader і FilterWriter

Абстрактный класс **FilterReader** предназначен для чтения фильтрованных символьных потоков.

Подклассом **FilterReader** является класс **PushbackReader**, который позволяет возвращать символы обратно в поток, т.е. действует аналогично классу **PushbackInputStream**.

Абстрактный класс **FilterWriter** предназначен для записи символов в фильтрованный символьный поток.

Класи StringReader і StringWriter

Классы **StringReader** и **StringWriter** создают соответственно входной поток из строки и выходной поток в строку.

Класи PipedReader і PipedWriter

Классы **PipedReader** и **PipedWriter** выполняют для символьных потоков функции, аналогичные функциям классов **PipedInputStream** и **PipedOutputStream** для байтовых потоков, т.е. связывают входной и выходной потоки в разных вычислительных потоках.

Клас PrintWriter

Класс **PrintWriter** выводит форматированное представление объектов в текстовый выводной поток

Консольне введення

Ранее был рассмотрен вывод на дисплей с использованием стандартных выходных потоков **System.out**. Аналогичным образом можно вводить данные с клавиатуры, не создавая собственных потоковых объектов с помощью объекта **System.in**, являющегося экземпляром класса **InputStream**. Поскольку объект является экземпляром класса, ему доступны все методы этого класса.

Признаком окончания ввода с клавиатуры может служить ввод заданного символа или последовательности символов, а также нажатие клавиш **Ctrl+Z**.

Хотя можно обрабатывать входной поток с клавиатуры как байтовый, в Java 2 рекомендуется преобразовать вводимый байтовый поток в символьный, используя объект класса **InputStreamReader**, а затем преобразовать его в буферный ввод с использованием класса **BufferedReader**.

Чтение данных с клавиатуры можно организовать с помощью следующей последовательности операторов

```
byte buffer[] = new byte[255];          // Объявление буферной области
                                        длиной 255 байт
int bufferCount = 0;                    // Количество введенных байт
 …
try {                                   // Проверка чтения из буфера
bufferCount = System.in.read(buffer, 0, 255);   // Чтение из буфера
}
catch (Exception e)                     // Обработка исключения
{
String err = e.toString();              // Формирование строки сообщения
System.out.println(err);                // Вывод строки сообщения на дисплей
}
```

Клас RandomAccessFile

Классы потоков (включая и символьные потоки) выполняют либо ввод, либо вывод информации, но не то и другое одновременно. Однако иногда в приложениях необходимо иметь доступ к некоторому файлу и для чтения, и для записи, причем одновременно. В этом случае работу с файлом следует выполнять с помощью класса **RandomAccessFile**.

Он напоминает комбинацию классов **DataInputFile** и **DataOutputFile**, дополненную некоторыми специальными методами, предназначенными для выполнения операций эффективного поиска в файле.

Клас StreamTokenizer

При чтении символьных потоков очень часто требуется разбивать данные на отдельные слова.

Слово – это последовательность символов, отделенная от других слов пробелом или некоторым символом-разделителем. Для решения подобной задачи в Java существует специальный класс **StreamTokenizer**, конструктору которого в качестве параметра передается ссылка на входной символьный поток.

Свойства этого класса определяют следующие типы слов:

TT_EOL – конец строки

TT_EOF – конец файла

TT_NUMBER – число

TT_WORD – текстовое поле