```
            System.out.println("каталог "
                  + dir.getName() + " существует");
         File[] files = dir.listFiles();
         for(int i = 0; i < files.length; i++){
            Date date = new Date(files[i].lastModified());
            System.out.print("\n" + files[i].getPath()
                  + " \t| " + files[i].length() + "\t| "
                  + date.toString());
          //использовать toLocaleString() или toGMTString()
       }
      // метод listRoots() возвращает доступные корневые каталоги
      File root = File.listRoots()[1];
      System.out.printf("\n%s %,d из %,d свободно.",
root.getPath(),root.getUsableSpace(),root.getTotalSpace());
   }
}
```

В результате файл **FileTest2.java** будет очищен, а на консоль выведено:

```
FileTest2.java существует
Путь к файлу:     chapt09\FileTest2.java
Абсолютный путь:  D:\workspace\chapt09\FileTest2.java
Размер файла:     2091
Последняя модификация : Fri Mar 31 12:26:50 EEST 2006
Файл доступен для чтения:     true
Файл доступен для записи:     true
Файл удален:      true
Файл FileTest2.java создан
каталог learn существует
com\learn\bb.txt  | 9   | Fri Mar 24 15:30:33 EET 2006
com\learn\byte.txt| 8   | Thu Jan 26 12:56:46 EET 2006
com\learn\cat.gif | 670 | Tue Feb 03 00:44:44 EET 2004
C:\ 3 665 334 272 из 15 751 376 896 свободно.
```

У каталога как объекта класса **File** есть дополнительное свойство – просмотр списка имен файлов с помощью методов **list()**, **listFiles()**, **listRoots()**.

Байтовые и символьные потоки ввода/вывода

При создании приложений всегда возникает необходимость прочитать информацию из какого-либо источника и сохранить результат. Действия по чтению/записи информации представляют собой стандартный и простой вид деятельности. Самые первые классы ввода/вывода связаны с передачей и извлечением последовательности байтов.

Потоки ввода последовательности байтов являются подклассами абстрактного класса **InputStream**, потоки вывода – подклассами абстрактного класса **OutputStream**. Эти классы являются суперклассами для ввода массивов байтов, строк, объектов, а также для выбора из файлов и сетевых соединений. При работе с файлами используются подклассы этих классов соответственно

FileInputStream и **FileOutputStream**, конструкторы которых открывают поток и связывают его с соответствующим физическим файлом.

Для чтения байта или массива байтов используются абстрактные методы **read()** и **read(byte[] b)** класса **InputStream**. Метод возвращает **–1**, если достигнут конец потока данных, поэтому возвращаемое значение имеет тип **int**, не **byte**. При взаимодействии с информационными потоками возможны различные исключительные ситуации, поэтому обработка исключений вида **try-catch** при использовании методов чтения и записи является обязательной. В конкретных классах потоков ввода указанные выше методы реализованы в соответствии с предназначением класса. В классе **FileInputStream** данный метод читает один байт из файла, а поток **System.in** как встроенный объект подкласса **InputStream** позволяет вводить информацию с консоли. Абстрактный метод **write(int b)** класса **OutputStream** записывает один байт в поток вывода. Оба эти метода блокируют поток до тех пор, пока байт не будет записан или прочитан. После окончания чтения или записи в поток его всегда следует закрывать с помощью метода **close()**, для того чтобы освободить ресурсы приложения.

Поток ввода связывается с одним из источников данных, в качестве которых могут быть использованы массив байтов, строка, файл, «pipe»-канал, сетевые соединения и др. Набор классов для взаимодействия с перечисленными источниками приведен на рис. 9.1.

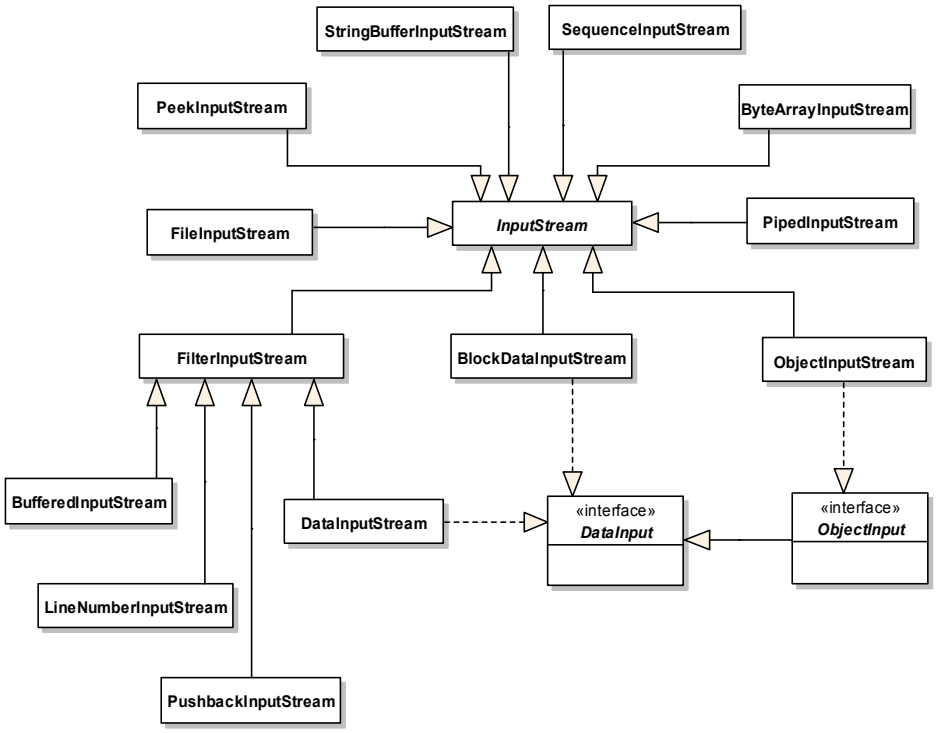

Рис. 9.1. Иерархия классов байтовых потоков ввода

Абстрактный класс **`FilterInputStream`** используется как шаблон для настройки классов ввода, наследуемых от класса **`InputStream`**. Класс **`DataInputStream`** предоставляет методы для чтения из потока данных значений базовых типов, но начиная с версии 1.2 класс был помечен как deprecated и не рекомендуется к использованию. Класс **`BufferedInputStream`** присоединяет к потоку буфер для ускорения последующего доступа.

Для вывода данных используются потоки следующих классов.

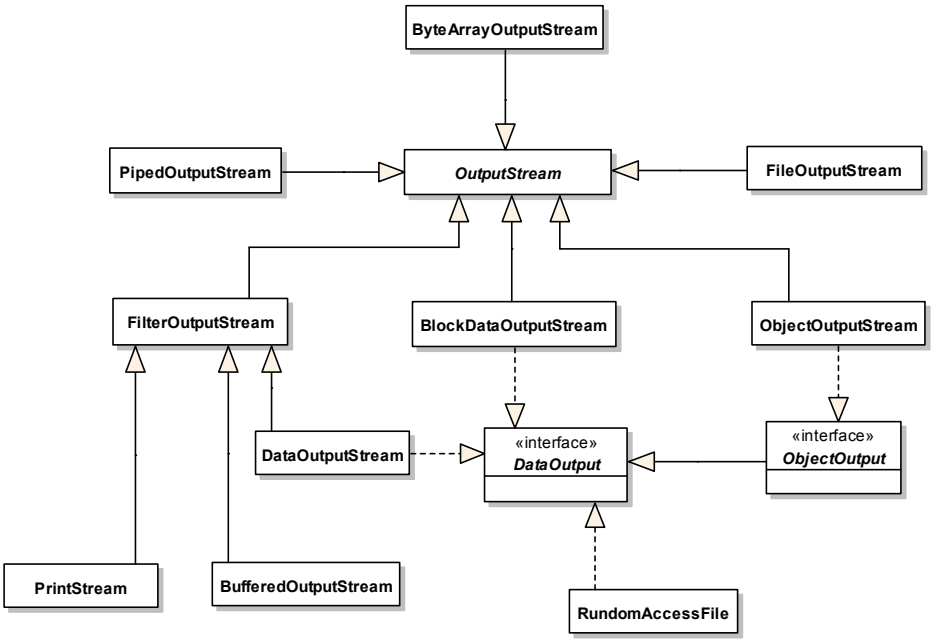

Рис. 9.2. Иерархия классов байтовых потоков вывода

Абстрактный класс **`FilterOutputStream`** используется как шаблон для настройки производных классов. Класс **`BufferedOutputStream`** присоединяет буфер к потоку для ускорения вывода и уменьшения доступа к внешним устройствам.

Начиная с версии 1.2 пакет **`java.io`** подвергся значительным изменениям. Появились новые классы, которые производят скоростную обработку потоков, хотя и не полностью перекрывают возможности классов предыдущей версии.

Для обработки символьных потоков в формате Unicode применяется отдельная иерархия подклассов абстрактных классов **`Reader`** и **`Writer`**, которые почти полностью повторяют функциональность байтовых потоков, но являются более актуальными при передаче текстовой информации. Например, аналогом класса **`FileInputStream`** является класс **`FileReader`**. Такой широкий выбор потоков позволяет выбрать наилучший способ записи в каждом конкретном случае.

В примерах по возможности используются способы инициализации для различных семейств потоков ввода/вывода.

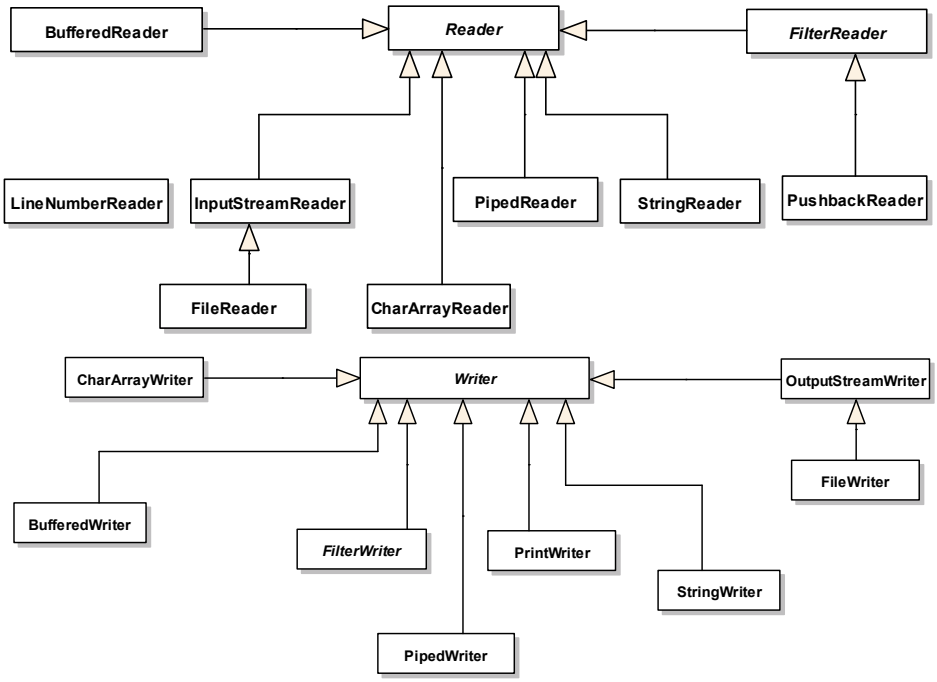

Рис. 9.3. Иерархия символьных потоков ввода/вывода

```
/* пример # 2 : чтение по одному байту (символу) из потока ввода : ReadDemo.java */
package chapt09;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class ReadDemo {
   public static void main(String[] args) {
      File f = new File("file.txt");//должен существовать!

      int b, count = 0;
      try {
         FileReader is = new FileReader(f);
      /* FileInputStream is = new FileInputStream(f);*/ //альтернатива
         while ((b = is.read()) != -1) {/*чтение*/
            System.out.print((char)b);
            count++;
         }
         is.close(); // закрытие потока ввода
       } catch (IOException e) {
            System.err.println("ошибка файла: " + e);
      }
      System.out.print("\n число байт = " + count);
   }
}
```

Один из конструкторов **FileReader(f)** или **FileInputStream(f)** открывает поток **is** и связывает его с файлом **f** . Для закрытия потока используется метод **close()**. При чтении из потока можно пропустить **n** байт с помощью метода **long skip(long n)**.

Для вывода символа (байта) или массива символов (байтов) в поток используются потоки вывода – объекты подкласса **FileWriter** суперкласса **Writer** или подкласса **FileOutputStream** суперкласса **OutputStream**. В следующем примере для вывода в связанный с файлом поток используется метод **write()**.

```
// пример # 3 : вывод массива в поток в виде символов и байтов: WriteRunner.java
package chapt09;
import java.io.*;

public class WriteRunner {
      public static void main(String[] args) {
            String pArray[] = { "2007 ", "Java SE 6" };
            File fbyte = new File("byte.txt");
            File fsymb = new File("symbol.txt");
            try {
                  FileOutputStream fos =
                        new FileOutputStream(fbyte);
                  FileWriter fw = new FileWriter(fsymb);
                  for (String a : pArray) {
                        fos.write(a.getBytes());
                        fw.write(a);
                  }
                  fos.close();
                  fw.close();
            } catch (IOException e) {
                  System.err.println("ошибка файла: " + e);
            }
      }
}
```

В результате будут получены два файла с идентичным набором данных, но созданные различными способами.

В отличие от классов **FileInputStream** и **FileOutputStream** класс **RandomAccessFile** позволяет осуществлять произвольный доступ к потокам как ввода, так и вывода. Поток рассматривается при этом как массив байтов, доступ к элементам осуществляется с помощью метода **seek(long poz)**. Для создания потока можно использовать один из конструкторов:

```
RandomAccessFile(String name, String mode);
RandomAccessFile(File file, String mode);
```

Параметр **mode** равен **"r"** для чтения или **"rw"** для чтения и записи.

```
/* пример # 4 : запись и чтение из потока: RandomFiles.java */
package chapt09;
import java.io.*;

public class RandomFiles {
```

```
    public static void main(String[] args) {
        double data[] = { 1, 10, 50, 200, 5000 };
        try {
        RandomAccessFile rf =
            new RandomAccessFile("temp.txt", "rw");
            for (double d : data)
                rf.writeDouble(d); // запись в файл
        /* чтение в обратном порядке */
        for (int i = data.length - 1; i >= 0; i--) {
                rf.seek(i * 8);
        // длина каждой переменной типа double равна 8-и байтам
            System.out.println(rf.readDouble());
            }
            rf.close();
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

В результате будет выведено:

```
5000.0
200.0
50.0
10.0
1.0
```

Предопределенные потоки

Система ввода/вывода языка Java содержит стандартные потоки ввода, вывода и вывода ошибок. Класс **System** пакета **java.lang** содержит поле **in**, которое является ссылкой на объект класса **InputStream**, и поля **out**, **err** – ссылки на объекты класса **PrintStream**, объявленные со спецификаторами **public static** и являющиеся стандартными потоками ввода, вывода и вывода ошибок соответственно. Эти потоки связаны с консолью, но могут быть переназначены на другое устройство.

Для назначения вывода текстовой информации в произвольный поток следует использовать класс **PrintWriter**, являющийся подклассом абстрактного класса **Writer**.

При наиболее удобного вывода информации в файл (или в любой другой поток) следует организовать следующую последовательность инициализации потоков с помощью класса **PrintWriter**:

```
new PrintWriter(new BufferedWriter(
      new FileWriter(new File("file.txt"))));
```

В итоге класс **BufferedWriter** выступает классом-оберткой для класса **FileWriter**, так же как и класс **BufferedReader** для **FileReader**.

Приведенный ниже пример демонстрирует вывод в файл строк и чисел с плавающей точкой.