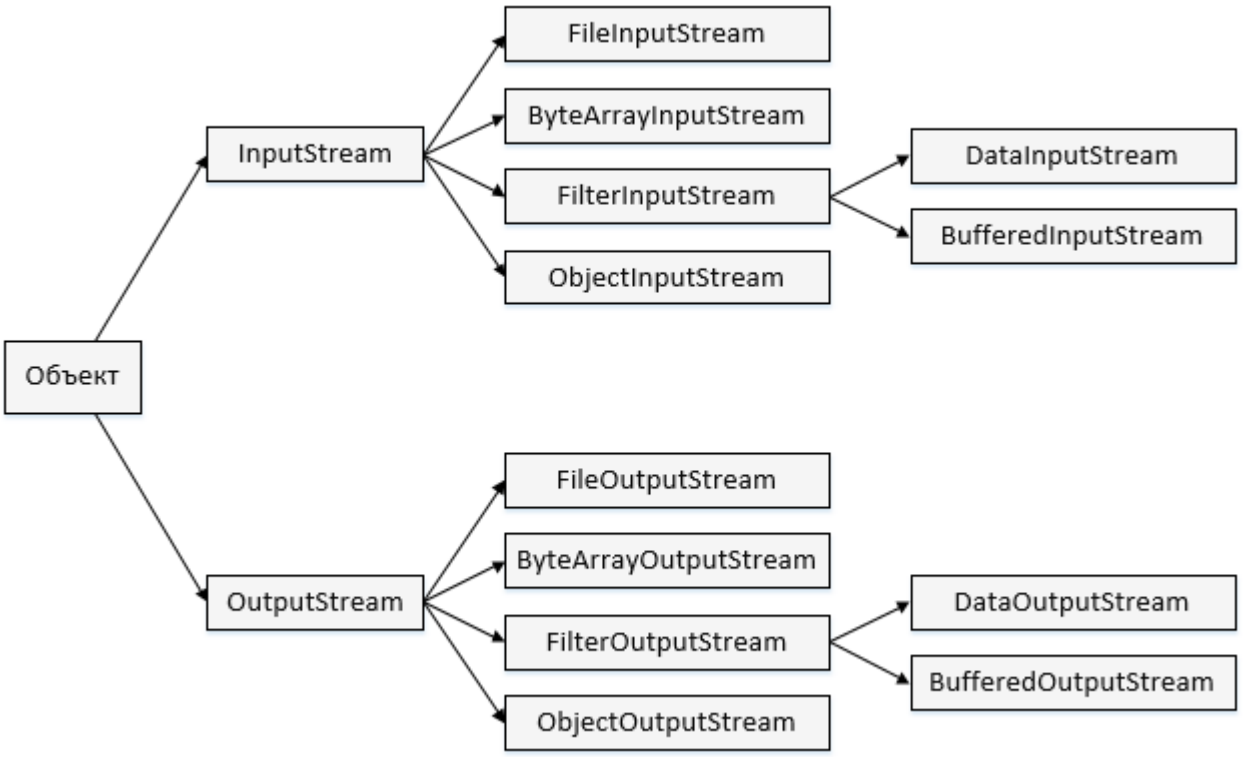
https://metanit.com/java/tutorial/6.4.php

Файловый ввод и вывод

Иерархия классов для управления потоками Ввода и Вывода



Байтовый поток

Потоки байтов в Java используются для осуществления ввода и вывода байтов. Наиболее распространено использование следующих классов:

FileInputStream **FileOutputStream**

Классы FileInputStream и FileOutputStream предназначены прежде всего для записи двоичных файлов, то есть для записи и чтения байтов. И хотя они также могут использоваться для работы с текстовыми файлами, но все же для этой задачи больше подходят другие классы.

Пример 1. Запись строки в файл

Для создания объекта **FileOutputStream** используется конструктор, принимающий в качестве параметра путь к файлу для записи. Если такого файла нет, то он автоматически создается при записи. Так как здесь записываем строку, то ее надо сначала перевести в массив байтов. И с помощью метода write строка записывается в файл.

Для автоматического закрытия файла и освобождения ресурса объект **FileOutputStream** создается с помощью конструктции **try...catch**.

Файл **Main.java**

```
import java.io.*;

public class Main {
    public static void main(String... args) {

        String text = "Hello world!"; // строка для записи
        try(FileOutputStream fos=new
FileOutputStream("d:\\notes.txt"))
        {
            // перевод строки в байты
            byte[] buffer = text.getBytes();

            fos.write(buffer, 0, buffer.length);
            fos.write(buffer[0]); // запись первого байта
        }
        catch(IOException ex){

            System.out.println(ex.getMessage());
        }
        System.out.println("The file has been written");
    }
}
```

Используя перегрузку метода write(), можно записать и одиночный байт:

```
fos.write(buffer[0]); // запись первого байта
```

Пример 2. Считывание данных по одному байту из ранее записанного файла и вывод на консоль

В данном случае мы считываем каждый отдельный байт в переменную i:. Когда в потоке больше нет данных для чтения, метод возвращает число -1. Затем каждый считанный байт конвертируется в объект типа char и выводится на консоль.

Файл **Main.java**

```
import java.io.*;


public class Main {
    public static void main(String... args) {


        try(FileInputStream fin=new
FileInputStream("d:\\notes.txt"))
        {
            System.out.printf("File size: %d bytes \n",
fin.available());

            int i=-1;
            while((i=fin.read())!=-1){

                System.out.print((char)i);
            }
        }
        catch(IOException ex){

            System.out.println(ex.getMessage());
        }
    }
}
```

Пример 3. Считывание данных в буфер целиком из ранее записанного файла и вывод на консоль

```
import java.io.*;


public class Main {
    public static void main(String... args) {


        try (FileInputStream fin = new
FileInputStream("d:\\notes.txt")) {
            byte[] buffer = new byte[fin.available()];
               // считаем файл в буфер
               fin.read(buffer, 0, fin.available());

            System.out.println("File data:");
            for (int i = 0; i < buffer.length; i++) {

                System.out.print((char) buffer[i]);
            }
        } catch (IOException ex) {

            System.out.println(ex.getMessage());
        }
    }
}
```

Пример 4. Чтение из одного и запись в другой файл

```
import java.io.*;

public class Main {
    public static void main(String... args) {

        try(FileInputStream fin=new
FileInputStream("d:\\notes.txt");

            FileOutputStream fos=new
FileOutputStream("d:\\notes_new.txt"))
        {
            byte[] buffer = new byte[fin.available()];
```

```
            // считываем буфер
            fin.read(buffer, 0, buffer.length);

            // записываем из буфера в файл
            fos.write(buffer, 0, buffer.length);
        }
        catch(IOException ex){

            System.out.println(ex.getMessage());
        }
    }
}
```

Закрытие потоков

При закрытии потока освобождаются все выделенные для него ресурсы, например, файл. В случае, если поток окажется не закрыт, может происходить утечка памяти.
Есть два способа закрытия файла. Первый традиционный заключается в использовании блока try..catch..finally.

Пример 4. Закрытие потоков. Первый способ. try..catch..finally
Поскольку при открытии или считывании файла может произойти ошибка ввода-вывода, то код считывания помещается в блок try.
И чтобы быть уверенным, что поток в любом случае закроется, даже если при работе с ним возникнет ошибка, вызов метода close() помещается в блок finally. И, так как метод close() также в случае ошибки может генерировать исключение IOException, то его вызов также помещается во вложенный блок try..catch.

```
import java.io.*;

public class Main {
    public static void main(String... args) {

        FileInputStream fin = null;
        try {
            fin = new FileInputStream("d:\\notes.txt");
```

```
            int i = -1;
            while ((i = fin.read()) != -1) {

                System.out.print((char) i);
            }
        } catch (IOException ex) {

            System.out.println(ex.getMessage());
        } finally {
            try {

                if (fin != null)
                    fin.close();
            } catch (IOException ex) {

                System.out.println(ex.getMessage());
            }
        }
    }
}
```

Пример 5. Закрытие потоков. Второй способ. try-with-resources

Начиная с Java 7 можно использовать еще один способ, который автоматически вызывает метод close. Этот способ заключается в использовании конструкции **try-with-resources (try-с-ресурсами)**.

Синтаксис конструкции следующий:

try(название_класса имя_переменной = конструктор_класса)

Данная конструкция также не исключает использования блоков catch.

После окончания работы в блоке **try** у ресурса (в данном случае у объекта FileInputStream) автоматически вызывается метод close().

```
import java.io.*;

public class Main {
    public static void main(String... args) {

        try(FileInputStream fin=new
FileInputStream("d:\\notes.txt"))
        {
```

```
            int i=-1;
            while((i=fin.read())!=-1){

                System.out.print((char)i);
            }
        }
        catch(IOException ex){

            System.out.println(ex.getMessage());
        }
    }
}
```

Если нам надо использовать несколько потоков, которые после выполнения надо закрыть, то мы можем указать объекты потоков через точку с запятой:

```
try(FileInputStream fin=new FileInputStream("d:\\file1.txt");
        FileOutputStream fos = new
FileOutputStream("d:\\file2.txt"))
        {
        //.....................
        }
```

Задания

1. С помощью клавиатуры ввести 10 чисел (по одному в каждую строку) (как строки) и записывать их в файл **task_01.txt**.

```
// подключить
import java.io.*;
import java.util.Scanner;

…
Scanner in = new Scanner(System.in);
…
System.out.print("Input n: ");
String st = in.nextLine();

// перевод строки в байты
…
```

2. Записать в файл **task_02.txt** десять случайных чисел в диапазоне от 0 до 33. Каждое число записывать в новую строку (символ перехода на следующую строку "\n").

```
// Генерация случайного числа от 0 до 33
int n = toIntExact(round(random() * 33));
// перевод числа в строку
String s=Integer.toString(n);
// перевод строки в массив байт
byte[] buffer = s.getBytes();
```

3. Записать в 9 файлов (file01.txt – file09.txt) по одному произвольному числу в диапазоне от 0 до 100.

```
// генерация имени файла
String fn="file0"+Integer.toString(i)+".txt";
```

4. Записать через пробел в файл **task_04.txt** 10 случайных чисел в диапазоне от 1 до 100.

5. Записать в файл **task_05.txt** 10 случайных строк.

```
…
// генерация случайной строки
st+=(char)(toIntExact(round(random()*32 + 65)));
…
```

6. Считать файл **task_05.txt** по одному байту и вывести на экран.

7. Считать файл **task_05.txt** в массив байт и вывести на экран.