

Java

Тема: Массивы. Строки. Файлы

Курс «Основы программирования
на Java»

Автор: А. Е. Анисимов, УдГУ

Тема 3: Массивы. Строки. Файлы

А. Массивы

В. Строки

С. Файлы

А. Массивы

А. Массивы

Массив — ссылочный тип для хранения однотипных данных

- Элементы массивов индексируются с нуля
- Массивы в Java являются динамическими, поэтому создаются с помощью **new** или инициализации



Пример программы 3.01.

```
/* Пример 3.01.Массивы.*/
public class ArrayClass {
    public static void main(String[] args) {
        int n = 10;
        int a[] = new int[n];
        int[] b = new int[]{10,20,30,40,50};
        double[] c = {2.72, 3.14, 9.81};

        for(int i=0; i < n; i++)
            a[i] = i*i;

        System.out.println("Array a:");
        for(int x : a)
            System.out.printf(" " + x);
        System.out.println("");

        System.out.println("Array b:");
        for(int i = 0; i < b.length; i++)
            System.out.printf(" " + b[i]);

    }
}
```

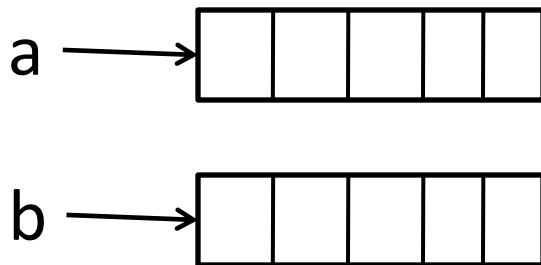
Console

```
Array a:
0 1 4 9 16 25 36 49 64 81
Array b:
10 20 30 40 50
```

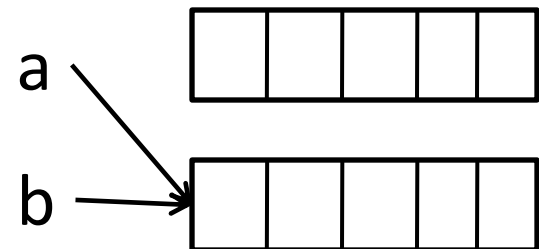
Пример программы 3.02.

```
/* Пример 3.02.  
   Сортировка массива */  
import java.util.*;  
public class SortArray {  
    public static void main(String[] args) {  
        Random random = new Random();  
        int n = 10;  
        int [] d = new int[n];  
        for(int i=0; i < d.length; i++)  
            d[i] = random.nextInt(100);  
  
        for(int r=n-2; r >=0; r--)  
            for(int i=0; i<=r; i++)  
                if(d[i] > d[i+1]) {  
                    int tmp = d[i];  
                    d[i] = d[i+1];  
                    d[i+1] = tmp;  
                }  
    }  
}
```

- Массив – это объект. У него есть
 - свойство **length** – количество его элементов
 - метод **clone()** – создает копию массива
- Элементами массива могут быть
 - значения базового типа
 - объекты
- Если значения не инициализированы, то
 - элементы базового типа будут иметь значения по умолчанию
 - объектные ссылки равны null
- Присваивание массивов вида **a = b** не будет переносом значений, так как это ссылки



a = b;



Пример программы 3.03.

```
/* Пример 3.03.  
   Массив объектов */  
public class ObjectArray {  
    public static void main(String[] args) {  
  
        Integer[] a = new Integer[3];  
        a[1] = new Integer(12);  
        for(Integer x : a)  
            System.out.println(""+ x);  
    }  
}
```

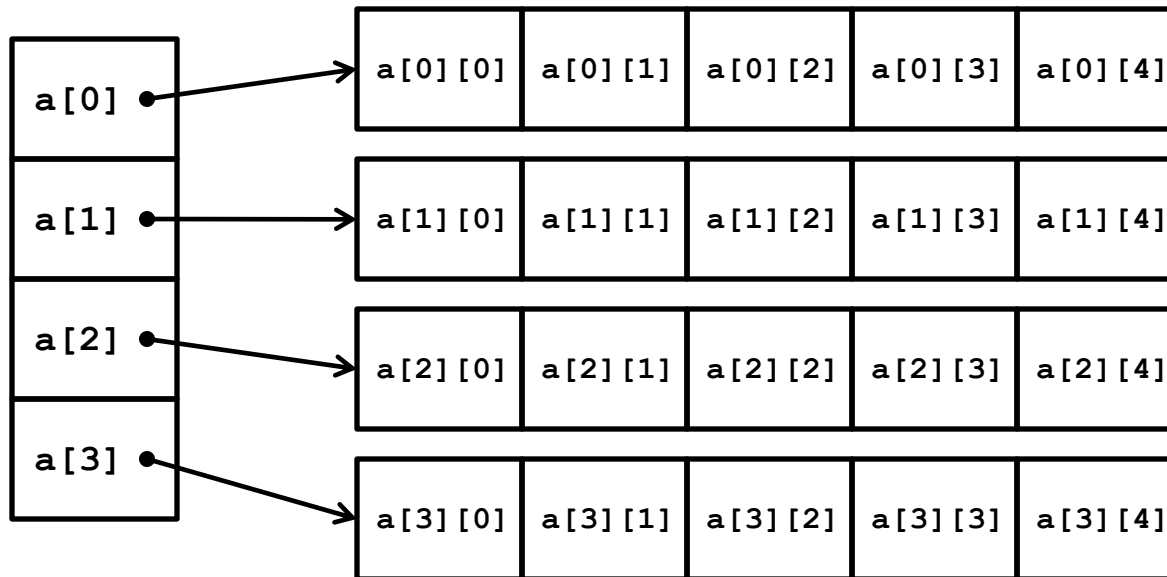
Console

```
null  
12  
null
```

Матрица – это двумерный массив

(на самом деле это массив массивов, то есть массив ссылок на массивы)

```
int a[][] = new int [4][5];
```



Пример программы 3.04.

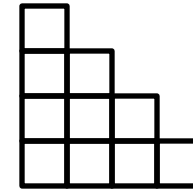
```
/* Пример 3.04. Матрица. Найти суммы по строкам */
import java.util.*;
public class Matrix{
    public static void main(String[] args) {
        Random random = new Random();
        int[][] a = new int[4][5];
        for(int i=0; i<a.length; i++){
            for(int j=0; j<a[i].length; j++){
                a[i][j] = random.nextInt(10);
                System.out.printf(" " + a[i][j]);
            }
            System.out.println("");
        }
        //Суммы по строкам
        for(int[] b: a){
            int s = 0;
            for(int c : b)
                s += c;
            System.out.println("Сумма в строке = "
                +s);
        }
    }
}
```

Console

```
5 2 1 6 2
5 8 2 9 8
7 3 9 1 3
7 7 5 5 4
Сумма в строке = 16
Сумма в строке = 32
Сумма в строке = 23
Сумма в строке = 28
```

Размеры массивов-строк матрицы могут отличаться друг от друга

```
// Треугольная матрица
int tryangle [][] = new int[4][];
tryangle[0] = new int [1];
tryangle[1] = new int [2];
tryangle[2] = new int [3];
tryangle[3] = new int [4];
// Инициализация матрицы
int [][] rhombus = {
    {1},
    {2, 3},
    {4, 5, 6},
    {7, 8},
    {9},
};
for(int i=0; i<rhombus.length; i++){
    for(int j=0; j<rhombus[i].length; j++){
        System.out.printf(" " + rhombus[i][j]);
    }
    System.out.println("");
}
```



Console

```
1
2 3
4 5 6
7 8
9
```

Пример программы 3.05.

```
/* Пример 3.05. Клонирование массивов */
public class CloneArray{
    public static void main(String[] args) {
        int[] a = {1, 2, 3} ;
        int[] b = a;
        int[] c = a.clone();

        b[0] = 555;
        c[0] = 999;

        System.out.println("Array a");
        for(int x: a)
            System.out.printf("  " + x);
        System.out.println();

        System.out.println("Array b");
        for(int x: b)
            System.out.printf("  " + x);
        System.out.println();

        System.out.println("Array c");
        for(int x: c)
            System.out.printf("  " + x);
        System.out.println();
    }
}
```

Console

```
Array a
  555  2  3
Array b
  555  2  3
Array c
  999  2  3
```

Обращение к несуществующему элементу массива отслеживается виртуальной машиной во время исполнения программы и порождает ошибку

```
public class ArrayIndexError {  
    public static void main(String[] args) {  
        int array[] = new int[] { 10, 20, 30 };  
        System.out.println(array[3]);  
    }  
}
```

Console

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:  
3 at ArrayClass.main(ArrayClass.java:4)
```

Класс Arrays - содержит ряд готовых методов для решения типовых задач с массивами, например

Arrays.sort(); Arrays.parallelSort();	Сортирует массив	int[] numbers = {2, 1, 5, 3, 7, 4}; Arrays.sort(numbers);
Arrays.toString();	Представляет массив в виде строки	Arrays.toString(numbers);
Arrays.copyOf();	Копирует массив	int[] copy = Arrays.copyOf(numbers, numbers.length);
Arrays.equals();	Сравнивает массивы	if (Arrays.equals(numbers, copy)) System.out.print("Yes");
Arrays.binarySearch()	Бинарный поиск в массиве	Arrays.binarySearch(numbers, key);

Задания к Теме 3: Массивы

Ввести с консоли n – размерность матрицы $a[n][n]$. Задать значения элементов матрицы в интервале значений от $-n$ до n с помощью датчика случайных чисел.

1. **ДЗ!** Найти в каждой строке матрицы наибольший из отрицательных элементов и из них найти наименьший.
2. **ДЗ!** Упорядочить строки (столбцы) матрицы в порядке возрастания значений элементов k -го столбца (строки).
3. **ДЗ!** Выполнить циклический сдвиг каждой строки заданной матрицы на k позиций вправо.
4. **ДЗ!** Найти сумму элементов матрицы, расположенных между первым и вторым положительными элементами каждой строки.

В. Строки

В. Строки

Строка — тип для хранения текстовой информации

- Строка — это не массив символов, а объект!
- Классы пакета `java.lang`:
 - **String**
 - **StringBuilder**
 - **StringBuffer**
- Для форматирования строк использую классы :
 - **Formatter**
 - **Pattern**
 - **Matcher** и др.

Особенности строковых классов:

String :

- используется по умолчанию
- после создания объекта его значение *не изменяется*; любое изменение приводит к созданию нового объекта

StringBuffer :

- значение объекта можно изменять
- потокобезопасный, в отличие от **StringBuilder**

StringBuilder :

- значение объекта можно изменять
- более эффективный, чем **StringBuffer**

Изменение значения объекта String

- Строка создается с помощью оператора **new()** или с помощью литерала; является объектом класса String

```
String str1 = new String("Ижевск");  
String str2 = "Москва";
```

- Значение объекта класса String не может быть изменено после создания объекта при помощи какого-либо метода класса
- Любое изменение строки приводит к созданию нового объекта.
- Ссылку на объект класса String можно изменить так, чтобы она указывала на другой объект и тем самым на другое значение

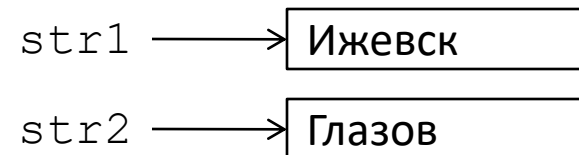
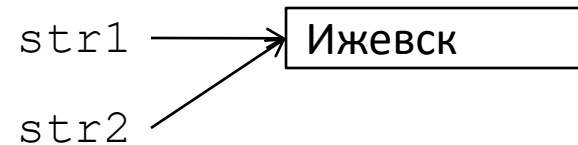
```
String str1 = new String("Ижевск");  
String str2 = str1;  
  
System.out.println("str1=" + str1 + " str2=" + str2);  
  
str2 = "Глазов";  
System.out.println("str1=" + str1 + " str2=" + str2);
```

Console

```
str1=Ижевск str2=Ижевск  
str1=Ижевск str2=Глазов
```

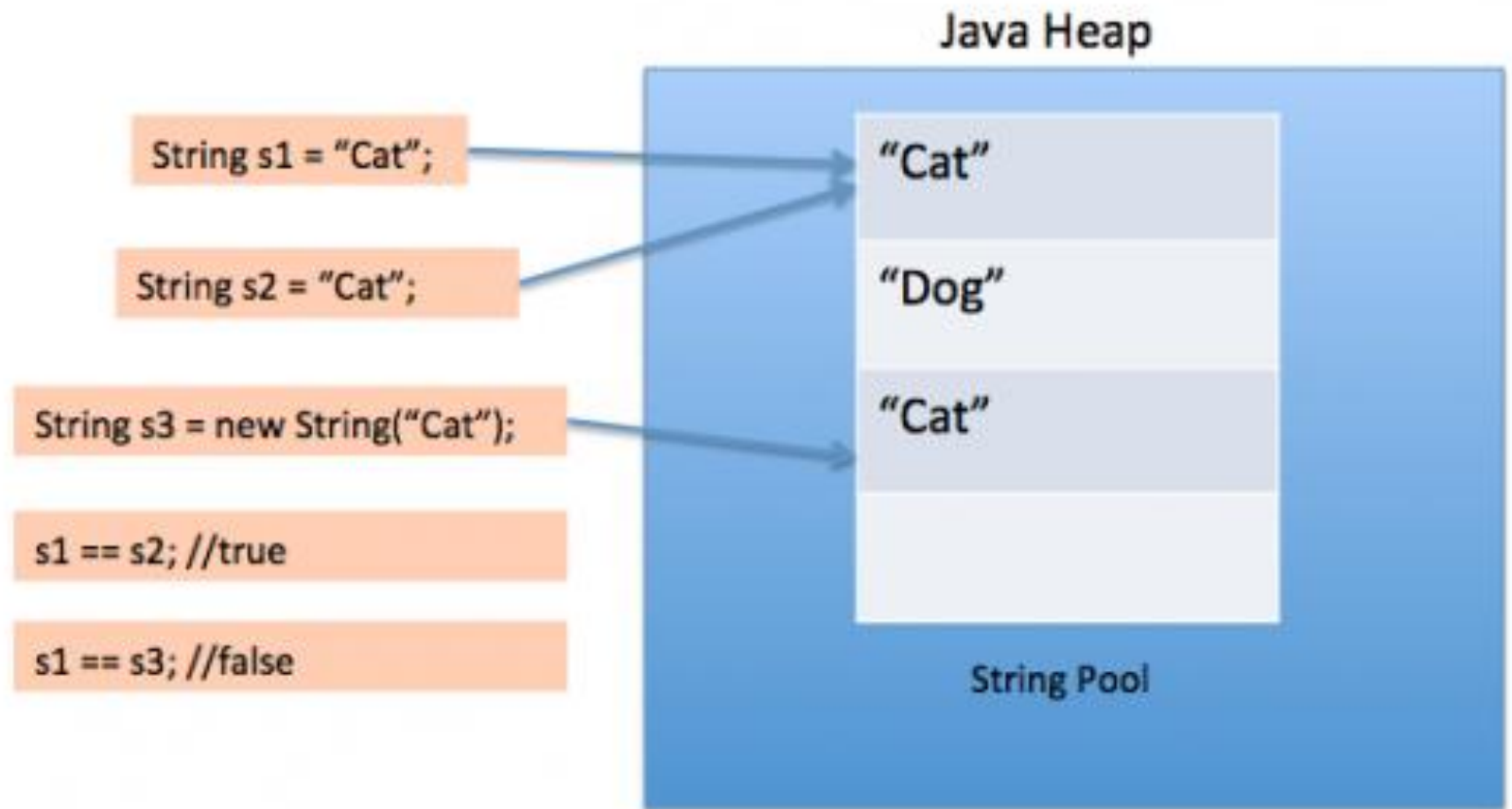
Ссылки

Литералы



Особенности хранения и идентификации строк

- В Java все ссылки хранятся в *стеке*, а объекты - в *куче*, строковые литералы сохраняются в *пуле строк*.
- В пул строк кладутся все строковые литералы, объявленные в коде: **String str = "Это литерал"** ;
- При совпадении литералов (в смысле equals) используется один и тот же объект, находящийся в пуле.
- Строку в пул можно поместить принудительно, с помощью метода `String.intern()`. Этот метод возвращает из пула строку, равную той, у которой был вызван этот метод. Если же такой строки нет – в пул кладется та, у которой вызван метод, после чего возвращается ссылка на нее же...



Сравнение строк. Сравнить значения строк на равенство необходимо с помощью методов

boolean equals(Object ob)

boolean equalsIgnoreCase(String s)

```
String str1 = new String("Ижевск");
String str2 = str1;

System.out.println(str1 == str2);      //true
System.out.println(str1.equals(str2)); //true

str1 = new String("Глазов");
str2 = new String("Глазов");

System.out.println(str1 == str2);      //false
System.out.println(str1.equals(str2)); //true
```

Console

```
true
true
false
true
```

Лексикографическое сравнение строк проводится с помощью методов

int compareTo(String s)

int compareToIgnoreCase(String s)

```
String str1 = "Я буду хорошим программистом!";  
String str2 = "Я буду хорошим программистом!";  
String str3 = "Я буду хорошим дворником!";  
String str4 = "Я буду ХОРОШИМ ДВОРНИКОМ!";  
  
System.out.println(str1.compareTo(str2)); //0, str1 equals str2  
System.out.println(str2.compareTo(str3)); // 11, str2 > str3  
System.out.println(str3.compareTo(str1)); // -11, str3 < str1  
System.out.println(str3.compareToIgnoreCase(str4)); // 0
```

Console

```
0  
11  
-11  
0
```

Некоторые операции и методы обработки строк String (1)

№	Методы	Описания
1	<code>char charAt(int index)</code>	Возвращает символ по указанному индексу.
2	<code>int compareTo(String anotherString)</code>	Сравнивает две строки лексически
3	<code>int compareToIgnoreCase(String str)</code>	Сравнивает две строки лексически, игнорируя регистр букв.
4	<code>String concat(String str)</code>	Объединяет указанную строку с данной строкой, путем добавления ее в конце (то же, что и +)
5	<code>String copyValueOf(char[] data)</code>	Возвращает строку, которая представляет собой последовательность символов, в указанный массив.
6	<code>boolean endsWith(String suffix)</code>	Проверяет заканчивается ли эта строка указанным окончанием
7	<code>boolean equalsIgnoreCase(String anotherString)</code>	Сравнивает данную строку с другой строкой, игнорируя регистр букв
8	<code>int hashCode()</code>	Возвращает хэш-код для этой строки.
9	<code>int indexOf(String str)</code>	Возвращает индекс первого вхождения указанной подстроки в данной строке
10	<code>int length()</code>	Возвращает длину строки.

Некоторые операции и методы обработки строк String (2)

№	Методы	Описания
11	boolean matches (String regex)	Сообщает, соответствует ли или нет эта строка заданному регулярному выражению.
12	String replace (char oldChar, char newChar)	Возвращает новую строку, в результате, заменив все вхождения oldChar в этой строке на newChar.
13	boolean startsWith (String prefix)	Проверяет, начинается ли эта строка с заданного префикса
14	String substring (int beginIndex)	Возвращает новую строку, которая является подстрокой данной строки
15	char[] toCharArray ()	Преобразует эту строку в новый массив символов
16	String toLowerCase ()	Преобразует все символы в данной строке в нижний регистр, используя правила данного языкового стандарта
17	String toUpperCase ()	Преобразует все символы в строке в верхний регистр, используя правила данного языкового стандарта.
18	String trim ()	Возвращает копию строки с пропущенными начальными и конечными пробелами
19	static String valueOf (primitive data type x)	Возвращает строковое представление переданного типа данных аргумента

Пример программы 3.06.

```
/* Пример 3.06. Проверка строки на палиндромность*/
import java.util.Scanner;
public class RefString {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите строку");
        String str = scanner.nextLine();
        int len = str.length();

        boolean isPalindrome = true;
        for (int i = 0; i < len / 2; i++) {
            if (str.charAt(i) != str.charAt(len - i - 1)){
                isPalindrome = false;
                break;
            }
        }
        if (isPalindrome)
            System.out.println("Это палиндром");
        else
            System.out.println("Это не палиндром");

        scanner.close();
    }
}
```

Console

Введите строку
мадам
Это палиндром

Console

Введите строку
лопата
Это не палиндром

Пример программы 3.07.

```
/* Пример 3.07. Удаление всех вхождений подстроки  
в строку*/
```

```
. . .  
System.out.println("Введите строку");  
String str = scanner.nextLine();  
  
System.out.println("Введите удаляемую подстроку");  
String str1 = scanner.nextLine();  
  
str = str.replaceAll(str1, "");  
System.out.println(str);  
. . .
```

Console

Введите строку

барабан

Введите удаляемую подстроку

ба

Полученная строка = ран

Пример программы 3.08.

```
/* Пример 3.08. Разбиение строки на части*/
```

```
. . .  
String str = new String("Ижевск столица Удмуртской Республики");  
String[] words = str.split(" ");  
for(String s : words)  
    System.out.println(s);  
. . .
```

Console

```
Ижевск  
столица  
Удмуртской  
Республики
```

Класс StringBuffer

Строки **String** являются неизменными, поэтому частая их модификация приводит к созданию новых объектов, что в свою очередь, приводит к расходу памяти. Для решения этой проблемы был создан класс `java.lang.StringBuffer`, который позволяет более эффективно работать над модификацией строки.

StringBuffer — близнец класса `String`, предоставляющий многое из того, что обычно требуется при работе со строками. Его объекты представляют собой последовательности символов, которые могут расширяться и модифицироваться.

Создание объектов **StringBuffer**

При создании объекта класса **StringBuffer**. Каждый объект имеет свою вместимость (*capacity*), которая отвечает за длину внутреннего буфера; по умолчанию *capacity* равна 16 символам. Если длина строки, что хранится в внутреннем буфере, не превышает *capacity*, то нет необходимости выделять новый массив буфера. Если же буфер переполняется — он автоматически становится больше.

Существует четыре способа создания объекта **StringBuffer**:

```
StringBuffer strB1 = new StringBuffer(); // capacity = 16
StringBuffer strB2 = new StringBuffer("Ижевск"); // capacity =
// длина строки + 16
StringBuffer strB3 = new StringBuffer(strB2); // параметр - любой
// класс, что реализует CharSequence
StringBuffer strB4 = new StringBuffer(50); // задаем capacity
```

Некоторые методы обработки строк StringBuffer

№	Методы	Описания
1	StringBuffer append (String str)	Присоединяет аргумент str к строке
2	StringBuffer delete (int start, int end)	Удаляет из строки символы от start до end
3	int indexOf (String str)	Возвращает индекс первого вхождения подстроки
4	StringBuffer insert (int offset, String str)	Вставка в строку строки str в позицию offset
5	int length ()	Возвращает длину строки
6	StringBuffer reverse ()	Переворачивает строку
7	StringBuffer substring (int start, int end)	Возвращает подстроку от start до end
8	и др...	

Пример программы 3.09.

```
/* Пример 3.09. Использование StringBuffer*/
public class RefString {
    public static void main(String[] args) {
        StringBuffer strB = new StringBuffer("Ижевск столица");
        System.out.println(strB);

        strB.append(" Удмуртии");
        System.out.println(strB);

        strB.delete(strB.indexOf("Ижевск"), "Ижевск".length());
        System.out.println(strB);

        strB.insert(0, "Глазов");
        System.out.println(strB);

        strB.insert(strB.indexOf("столица"), "прежняя ");
        System.out.println(strB);
    }
}
```

Console

```
Ижевск столица
Ижевск столица Удмуртии
столица Удмуртии
Глазов столица Удмуртии
Глазов прежняя столица Удмуртии
```

Класс `StringBuilder`

`StringBuilder` — был введен в Java 5 и имеет полностью идентичный интерфейс с **`StringBuffer`**. Единственное отличие — `StringBuilder` не синхронизирован. Это означает, что его использование в многопоточных средах нежелательно.

Следовательно, если приложение работает с многопоточностью, то для него идеально подходит `StringBuffer`, иначе лучше использовать `StringBuilder`, который работает намного быстрее в большинстве случаев.

Класс Pattern

(в разработке)

Задания к Теме 3: Строки

Ввод строк производить с клавиатуры (консоли).

1. Дана строка символов. Удалить из строки самое длинное слово.
2. Дана строка символов. Каждое четное по номеру слово строки преобразовать в верхний регистр, каждое нечетное – в нижний.
3. Дана строка символов. Построить новую строку, разместив в ней слова данной строки в алфавитном порядке.
4. Дана строка символов. Удалить повторные вхождения слов строки.
5. Даны две строки символов. Удалить из первой строки слова, имеющиеся во второй строке.

С. Файлы

Библиотека ввода/вывода **java.io** предоставляет пользователю большое число классов и методов с файлами и потоками:

File – работа с файловой системой

InputStream, OutputStream – работа с потоками ввода и вывода

FileInputStream, FileOutputStream
работа с файлами как потоками ввода и вывода

Класс `java.io.File` :

- служит для хранения и обработки в качестве объектов каталогов и имен файлов
- не содержит методы для работы с содержимым файла, но позволяет манипулировать такими свойствами файла, как права доступа, дата и время создания, путь в иерархии каталогов, создание, удаление файла, изменение его имени и каталога и др.

Примеры создания объекта класса `File`:

```
File myFile1 = new File("\\com\\myfile.txt");  
File myFile2 = new File("c:\\com", "myfile.txt");  
File myFile3 = new File(new URI("Интернет-адрес"));
```

Примечание. 1. Двойной символ "\\" обозначает на самом деле одиночный символ "\", так как обратная наклонная черта в строках трактуется компилятором как управляющий символ.

2. Пример приведен для файловой системы Windows

3. При создании объекта класса `File` не выполняется проверка на существование физического файла с заданным путем.

Разделители в записи пути к файлу

- Существует разница между разделителями, употребляющимися при записи пути к файлу: для системы Unix – “/”, а для Windows – “\\”.
- Для случаев, когда неизвестно, в какой системе будет выполняться код, предусмотрены специальные поля в классе File:

```
public static final String separator;  
public static final char separatorChar;
```

- С помощью этих полей можно задать путь, универсальный в любой системе, например:

```
File myFile = new File(File.separator + "com" +  
    File.separator + "myfile.txt" );
```

- Также предусмотрен еще один тип разделителей – для директорий:

```
public static final String pathSeparator;  
public static final char pathSeparatorChar;
```

В классе **File** объявлено более тридцати методов, наиболее используемые из них рассмотрены в следующем примере

Пример программы 3.10...

```
/* Пример 3.10.  Использование File*/
import java.io.*;
import java.util.*;

public class FileTest {
    public static void main(String[] args)
    {
        //с объектом типа File ассоциируется файл на диске FileTest2.java
        File fp = new File("FileTest2.java");
        try{
            if(fp.createNewFile())
                System.out.println("Файл " + fp.getName() + " создан");
        }catch(IOException e)
        { System.err.println(e); }

        if(fp.exists()) {
            System.out.println(fp.getName() + " существует");
            // см. далее ->
        }
    }
}
```

Пример программы ..3.10...

```
if(fp.isFile()) {//объект - дисковый файл
    System.out.println("Путь к файлу:\t" + fp.getPath());
    System.out.println("Абсолютный путь:\t" + fp.getAbsolutePath());
    System.out.println("Размер файла:\t" + fp.length());
    System.out.println("Последняя модификация :\t" + new Date(fp.lastModified()));
    System.out.println("Файл доступен для чтения:\t" + fp.canRead());
    System.out.println("Файл доступен для записи:\t" + fp.canWrite());
}
else
    System.out.println("файл " + fp.getName() + " не существует");

// объект - каталог. Вывод на печать содержимого каталога данного проекта
File dir = new File("..\" + File.separator + "FileTest");
if (dir.exists() && dir.isDirectory())
    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());
}
}
```

Пример программы ..3.10.

Console

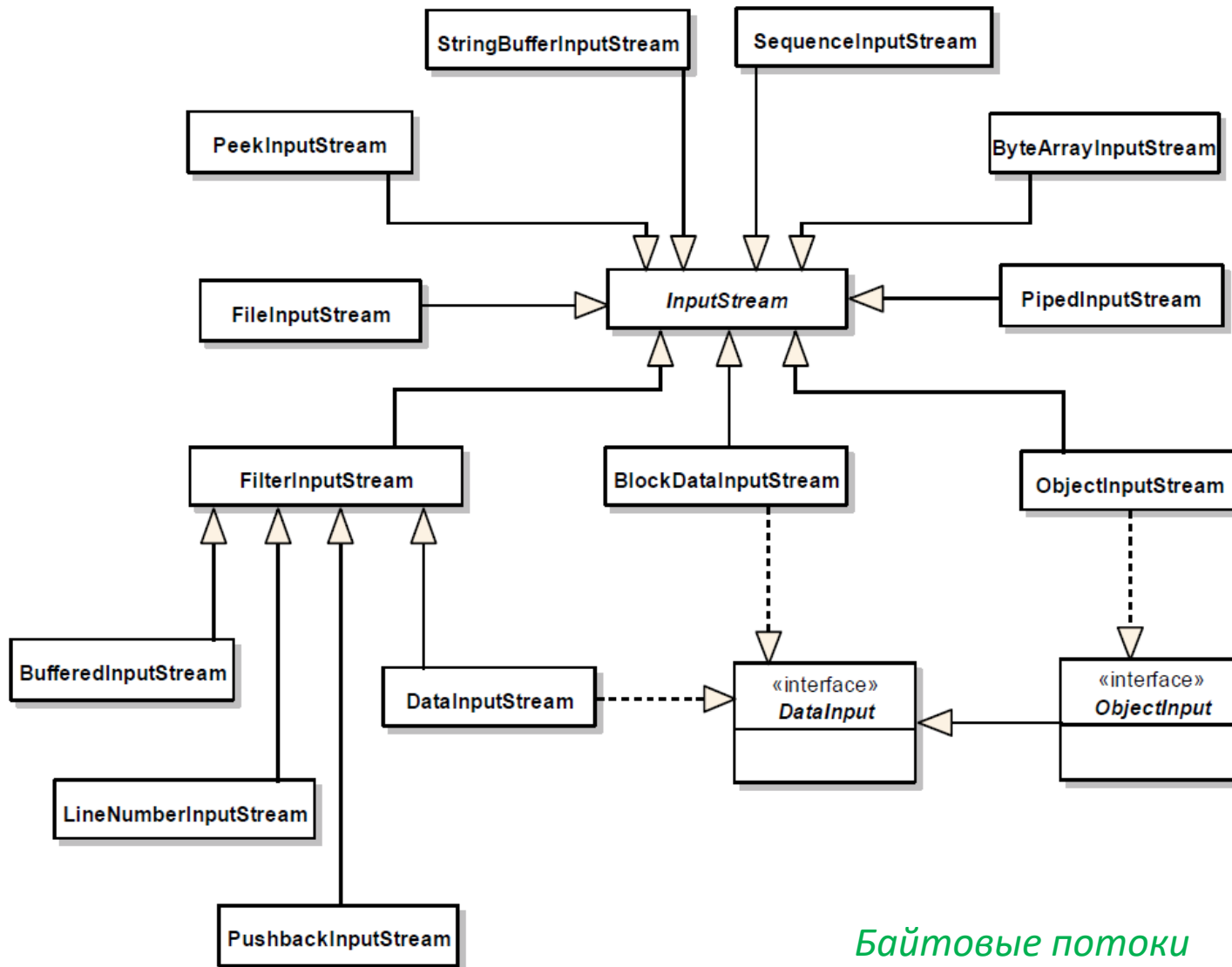
```
FileTest2.java существует
Путь к файлу:FileTest2.java
Абсолютный путь:C:\Users\Anisimov\workspace\FileTest\FileTest2.java
Размер файла:0
Последняя модификация :Sat Jan 21 03:34:20 BRT 2017
Файл доступен для чтения:true
Файл доступен для записи:true
каталог FileTest существует

..\FileTest\classpath          | 301| Sat Jan 21 02:50:20 BRT 2017
..\FileTest\project            | 384| Sat Jan 21 02:50:20 BRT 2017
..\FileTest\settings           | 0  | Sat Jan 21 02:50:20 BRT 2017
..\FileTest\bin                | 0  | Sat Jan 21 02:50:36 BRT 2017
..\FileTest\FileTest2.java     | 0  | Sat Jan 21 03:34:20 BRT 2017
..\FileTest\src                | 0  | Sat Jan 21 02:50:36 BRT 2017
```

Класс `java.io.InputStream/OutputStream`

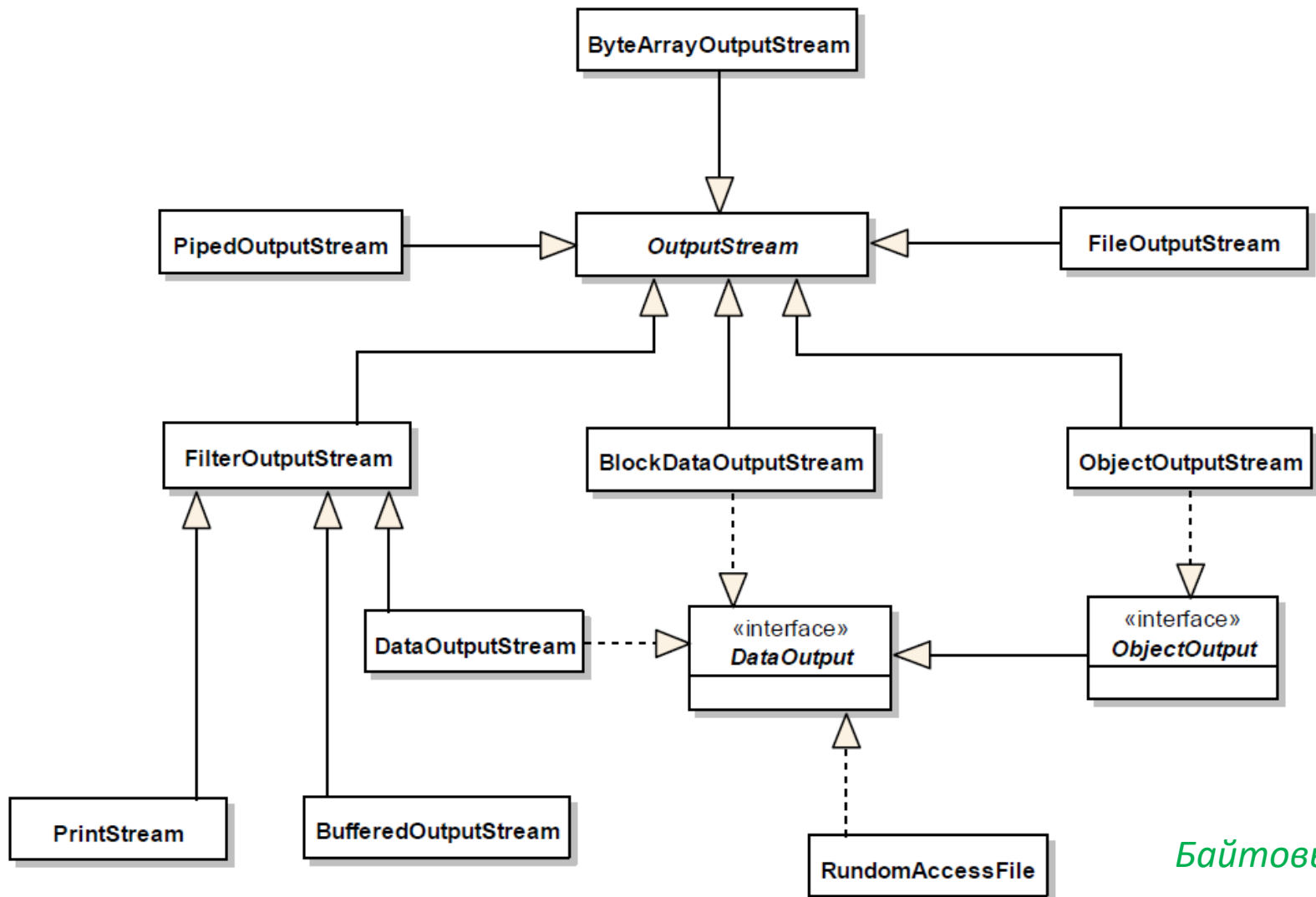
- При создании приложений всегда возникает необходимость прочесть информацию из какого-либо источника и сохранить результат. Действия по чтению/записи информации представляют собой стандартный и простой вид деятельности. Самые первые классы ввода/вывода связаны с передачей и извлечением последовательности байтов.
- **Поток данных** (data stream) - абстракция источника/приемника данных, унифицирующая операции для различных видов источников (консоль, файлы, сокет, процессы и др.)

Иерархия классов потоков ввода



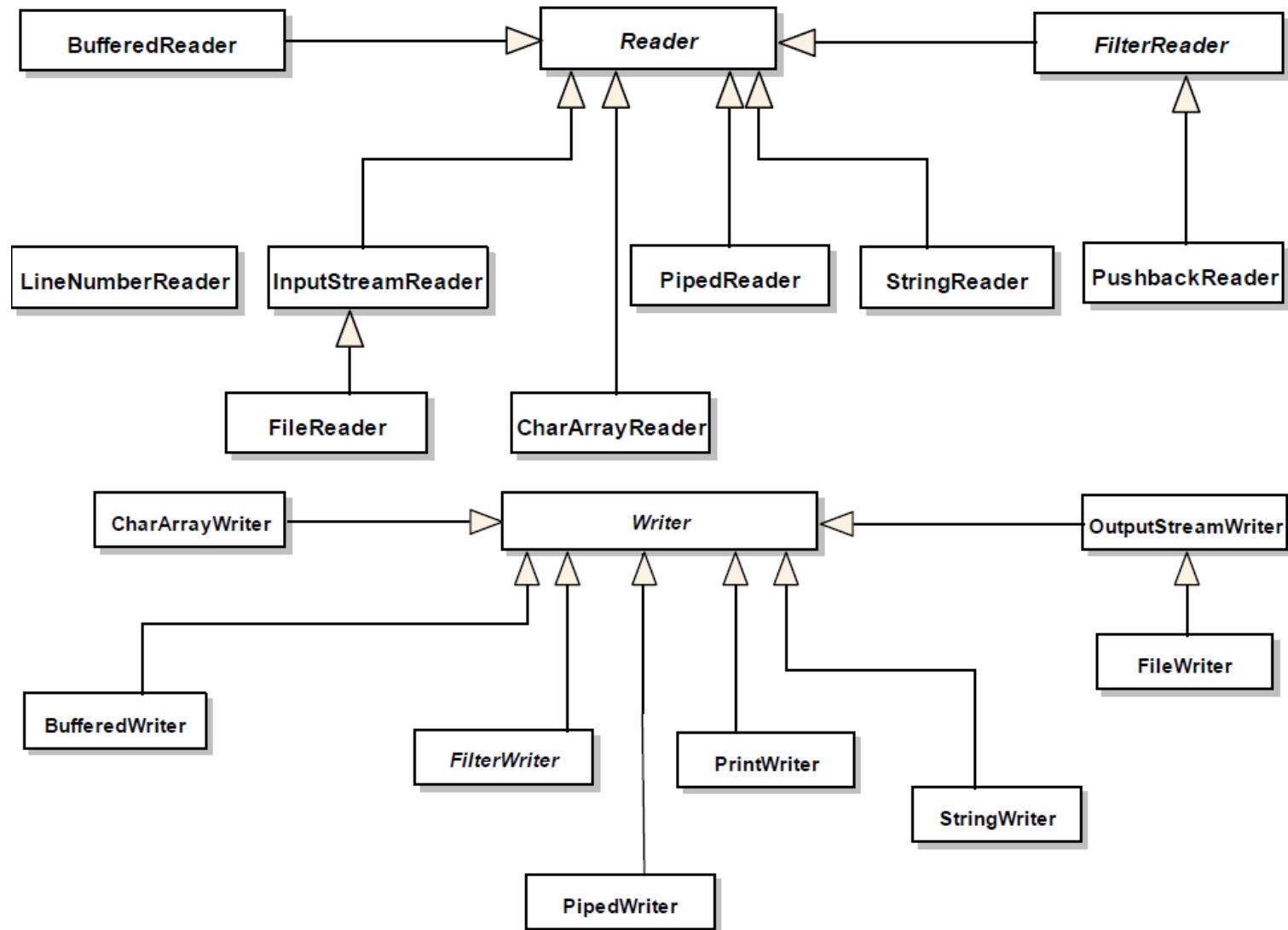
Байтовые потоки

Иерархия классов потоков вывода



Байтовые потоки

Иерархия строковых потоков ввода и вывода



Для чтения байта или массива байтов используются 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 классе **FileInputStream** данный метод читает один байт из файла, а поток **System.in** как встроенный объект подкласса **InputStream** позволяет вводить информацию с консоли.

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.

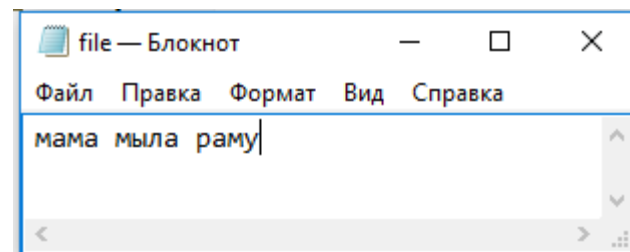
После окончания чтения или записи в поток его всегда следует закрывать с помощью метода **close()**, для того чтобы освободить ресурсы приложения. Это делается в блоке **finally**

Пример программы 3.11.

```
/* Пример 3.11. чтение по одному байту (символу) из потока ввода */
import java.io.*;
public class FileTest3 {
    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);
    }
}
```

Console

мама мыла раму
число байт = 14



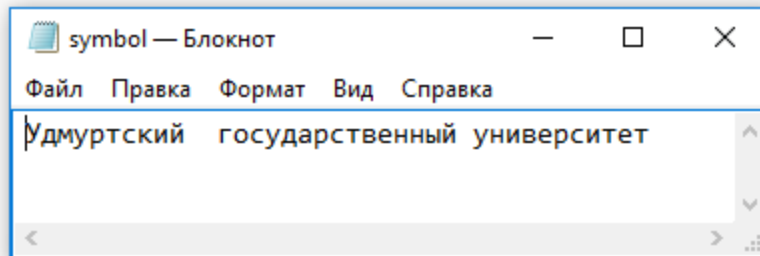
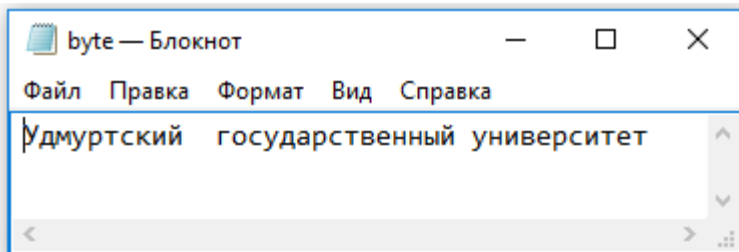
Для вывода символа (байта) или массива символов (байтов) в поток используются потоки вывода – объекты подкласса **FileWriter** суперкласса **Writer** или подкласса **FileOutputStream** суперкласса **OutputStream**.

В следующем примере для вывода в связанный с файлом поток используется метод **write()**.

Пример программы 3.12.

```
/* Пример 3.12.вывод в поток: два способа */
import java.io.*;
public class FileTest4 {
    public static void main(String[] args) {
        String pArray[] = { "Удмуртский", "  государственный",
                            " университет" };

        try {
            FileOutputStream fos = new FileOutputStream("byte.txt");
            FileWriter fw = new FileWriter("symbol.txt");
            for (String s : pArray) {
                fos.write(s.getBytes());
                fw.write(s);
            }
            fos.close();
            fw.close();
        } catch (IOException e) {
            System.err.println("ошибка файла: " + e);
        }
    }
}
```



Класс `java.util.Scanner`

- Объект класса **Scanner** принимает форматированный объект из входного потока и преобразует его в двоичное представление
- Инициализация объекта класса **Scanner** производится тем потоком, из которого будет ввод, например, стандартным потоком ввода **System.in**, объектом класса **File** или **InputStream**, и др.
- Из последовательности байтов **Scanner** формирует лексему и представляет в том типе, который запрошен соответствующим методом: **nextInt()**, **nextDouble()**, **nextBoolean()**, **nextFloat()**, **nextLine()** и т.д.
- Для проверки типа входного значения используют методы **hasNextInt()**, **hasNextDouble()** и т.д.

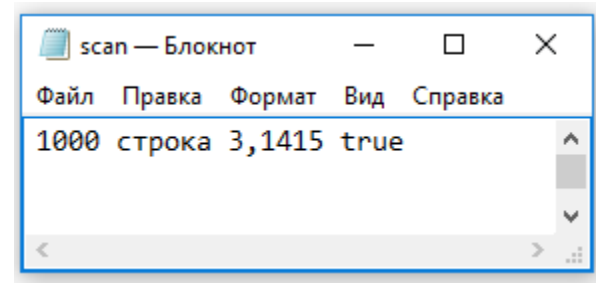
Java Тема: Массивы. Строки. Файлы

Пример программы 3.13.

```
/* Пример 3.13.Ввод из файлового потока, класс Scanner */
...
try {
    FileReader fr = new FileReader("scan.txt");
    Scanner scan = new Scanner(fr);
    // чтение из файла
    while (scan.hasNext()) {
        if (scan.hasNextInt())
            System.out.println(scan.nextInt() + " :int");
        else if (scan.hasNextDouble())
            System.out.println(scan.nextDouble() + " :double");
        else if (scan.hasNextBoolean())
            System.out.println(scan.nextBoolean() + " :boolean");
        else
            System.out.println(scan.next() + " :String");
    }
    scan.close();
} catch (FileNotFoundException e) {System.err.println(e);}
. . .
```

Console

```
1000 :int
строка :String
3.1415 :double
true :boolean
```



Предопределенные потоки

Система ввода/вывода содержит стандартные потоки, связанные с консолью:

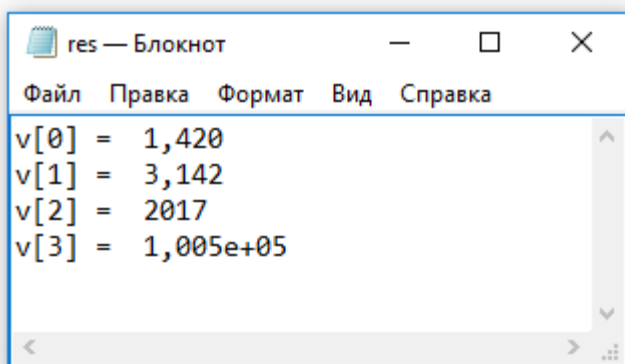
- **System.in** – поток ввода, ссылка на объект класса `InputStream`
- **System.out** - поток вывода, ссылка на объект класса `PrintStream`
- **System.err** - поток вывода ошибок, ссылка на объект класса `PrintStream`

Для 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"))));
```

Пример программы 3.14.

```
/* Пример 3.14.Вывод в файл с помощью FileWriter */
...
try {
File f = new File("res.txt");
FileWriter fw = null;
try {
    fw = new FileWriter(f, true);
} catch (IOException e) {
    System.err.println("ошибка открытия потока " + e);
    System.exit(1);
}
BufferedWriter bw = new BufferedWriter(fw);
PrintWriter pw = new PrintWriter(bw);
double[] v = { 1.42, 3.1415926, 2017, 100500};
for (int i=0; i < v.length; i++)
    pw.printf("v[%d] = %.4g%n", i, v[i]);
pw.close();
. . .
```



Пример программы 3.15.

```
/* Пример 3.15. Чтение и вывод в файл */
import java.io.*;
import java.util.*;
public class FileTest {
    public static void main(String[] args)
    {
        String[] names = new String[100];
        int[] marks = new int[100];
        int count = 0;
        try{
            FileReader fr = new FileReader("abitur.txt");
            Scanner scan = new Scanner(fr);

            while (scan.hasNext()){
                names[count] = scan.next();
                marks[count++] = scan.nextInt();
            }
            scan.close();
        } catch (FileNotFoundException e) {
            System.err.println(e);
        } // - продолжение ->
```

Пример программы 3.15.

```
for (int i = marks.length-1; i > 0; i--)
    for (int j = 0; j < i; j++)
        if(marks[j] < marks[j+1]){
            int t = marks[j];
            marks[j] = marks[j+1];
            marks[j+1] = t;
            String s = names[j];
            names[j] = names[j+1];
            names[j+1] = s;
        }
    try {        PrintWriter pw = new PrintWriter(new BufferedWriter(new
FileWriter(new File("abitur1.txt"), false)));
        for(int i=0; i < count; i++) {
            pw.println(names[i] + " " + marks[i]);
        }
        pw.close();
    } catch (IOException e) {
        System.err.println("ошибка открытия потока " + e);
        System.exit(1);
    }
}
```

Задания к Теме 3: Файлы

1. Создать и заполнить файл случайными целыми числами. Отсортировать содержимое файла по возрастанию.
2. В файле содержится последовательность строк. Каждая строка состоит из последовательности слов. Найти, сколько слов в каждой строке (начиная со второй) имеется в первой строке файла.
3. Прочитать текст Java-программы и все слова **public** заменить на слово **private**, сохранив новый текст в выходном файле.
4. Файл содержит символы, целые числа и числа с плавающей запятой. Определить количество значений каждого из типов.