```
            if(a[i].compareTo(a[j]) < 0)    {
                String temp = a[j];
                a[j] = a[i];
                a[i] = temp;
            }
        int i = -1;
        while(++i < a.length)
            System.out.print(a[i] + " ");
    }
}
```

Вызов метода **trim()** обеспечивает удаление всех начальных и конечных символов пробелов. Метод **compareTo()** выполняет лексикографическое сравнение строк между собой по правилам Unicode.

Классы StringBuilder и StringBuffer

Классы **StringBuilder** и **StringBuffer** являются "близнецами" и по своему предназначению близки к классу **String**, но, в отличие от последнего, содержимое и размеры объектов классов **StringBuilder** и **StringBuffer** можно изменять.

Основным и единственным отличием **StringBuilder** от **StringBuffer** является потокобезопасность последнего. В версии 1.5.0 был добавлен непотокобезопасный (следовательно, более быстрый в обработке) класс **StringBuilder**, который следует применять, если не существует вероятности использования объекта в конкурирующих потоках.

С помощью соответствующих методов и конструкторов объекты классов **StringBuffer**, **StringBuilder** и **String** можно преобразовывать друг в друга. Конструктор класса **StringBuffer** (также как и **StringBuilder**) может принимать в качестве параметра объект **String** или неотрицательный размер буфера. Объекты этого класса можно преобразовать в объект класса **String** методом **toString()** или с помощью конструктора класса **String**.

Следует обратить внимание на следующие методы:

void setLength(int n) – установка размера буфера;

void ensureCapacity(int minimum) – установка гарантированного минимального размера буфера;

int capacity() – возвращение текущего размера буфера;

StringBuffer append(параметры**)** – добавление к содержимому объекта строкового представления аргумента, который может быть символом, значением базового типа, массивом и строкой;

StringBuffer insert(параметры**)** – вставка символа, объекта или строки в указанную позицию;

StringBuffer deleteCharAt(int index) – удаление символа;

StringBuffer delete(int start, int end) – удаление подстроки;

StringBuffer reverse() – обращение содержимого объекта.

В классе присутствуют также методы, аналогичные методам класса **String**, такие как **replace()**, **substring()**, **charAt()**, **length()**, **getChars()**, **indexOf()** и др.

```
/* пример # 6 : свойства объекта StringBuffer: DemoStringBuffer.java */
package chapt07;

public class DemoStringBuffer {
     public static void main(String[] args) {
          StringBuffer sb = new StringBuffer();
          System.out.println("длина ->" + sb.length());
          System.out.println("размер ->" + sb.capacity());
          // sb = "Java"; // ошибка, только для класса String
          sb.append("Java");
          System.out.println("строка ->" + sb);
          System.out.println("длина ->" + sb.length());
          System.out.println("размер ->" + sb.capacity());
          System.out.println("реверс ->" + sb.reverse());
     }
}
```

Результатом выполнения данного кода будет:

```
длина ->0
размер ->16
строка ->Java
длина ->4
размер ->16
реверс ->avaJ
```

При создании объекта **StringBuffer** конструктор по умолчанию автоматически резервирует некоторый объем памяти (16 символов), что в дальнейшем позволяет быстро менять содержимое объекта, оставаясь в границах участка памяти, выделенного под объект. Размер резервируемой памяти при необходимости можно указывать в конструкторе. Если длина строки **StringBuffer** после изменения превышает его размер, то ёмкость объекта автоматически увеличивается, оставляя при этом резерв для дальнейших изменений. С помощью метода **reverse()** можно быстро изменить порядок символов в объекте.

Если метод, вызываемый объектом **StringBuffer**, производит изменения в его содержимом, то это не приводит к созданию нового объекта, как в случае объекта **String**, а изменяет текущий объект **StringBuffer**.

```
/* пример # 7 : изменение объекта StringBuffer: RefStringBuffer.java */
package chapt07;

public class RefStringBuffer {
     public static void changeStr(StringBuffer s) {
          s.append(" Microsystems");
     }
     public static void main(String[] args) {
          StringBuffer str = new StringBuffer("Sun");
          changeStr(str);
          System.out.println(str);
     }
}
```

В результате выполнения этого кода будет выведена строка:

```
Sun Microsystems
```

Объект **StringBuffer** передан в метод **changeStr()** по ссылке, поэтому все изменения объекта сохраняются и для вызывающего метода.

Для класса **StringBuffer** не переопределены методы **equals()** и **hashCode()**, т.е. сравнить содержимое двух объектов невозможно, к тому же хэш-коды всех объектов этого типа вычисляются так же, как и для класса **Object**.

```
/*пример # 8 : сравнение объектов StringBuffer и их хэш-кодов:
 EqualsStringBuffer.java */
package chapt07;

public class EqualsStringBuffer {
         public static void main(String[] args) {
         StringBuffer sb1 = new StringBuffer("Sun");
         StringBuffer sb2 = new StringBuffer("Sun");
         System.out.print(sb1.equals(sb2));
         System.out.print(sb1.hashCode() ==
                                    sb2.hashCode());
     }
}
```

Результатом выполнения данной программы будет дважды выведенное значение **false**.

Форматирование строк

Для создания форматированного текстового вывода предназначен класс **java.util.Formatter**. Этот класс обеспечивает преобразование формата, позволяющее выводить числа, строки, время и даты в любом необходимом разработчику виде.

В классе **Formatter** объявлен метод **format()**, который преобразует переданные в него параметры в строку заданного формата и сохраняет в объекте типа **Formatter**. Аналогичный метод объявлен у классов **PrintStream** и **PrintWriter**. Кроме того, у этих классов объявлен метод **printf()** с параметрами идентичными параметрам метода **format()**, который осуществляет форматированный вывод в поток, тогда как метод **format()** сохраняет изменения в объекте типа **Formatter**. Таким образом, метод **printf()** автоматически использует возможности класса **Fomatter** и подобен функции **printf()** языка С.

Класс **Formatter** преобразует двоичную форму представления данных в форматированный текст. Он сохраняет форматированный текст в буфере, содержимое которого можно получить в любой момент. Можно предоставить классу **Formatter** автоматическую поддержку этого буфера либо задать его явно при создании объекта. Существует возможность сохранения буфера класса **Formatter** в файле.

Для создания объекта класса существует более десяти конструкторов. Ниже приведены наиболее употребляемые:

Formatter()