

Collating Sequence

The bit patterns representing a character can be interpreted as an unsigned integer and so the natural order of numbers can be used to order the characters.

Collating sequence. The collating sequence of a character set is the order of the underlying bit representation. $c1 < c2$ is defined to be $((int) c1) < ((int) c2)$. In fact, the cast is a no-op in Java; the bits stay the same, only the interpretation changes.

Characters are automatically promoted to integers (no cast is needed). Characters are *not* automatically promoted to **short**. One can cast them to **short**; this is a bad idea even though no bits are lost, because the 16-bit, two's-complement representation of **short** is incompatible with the intuitive, collating sequence of 16-bit **char**.

```
final char xc = 'A';  
final char zc = '\ufb01'; // fi ligature  
final short xs = (short)xc, zs = (short)zc;  
System.out.println (xc<xz); // true  
System.out.println (xs<zs); // false
```

This is no unsigned, 16-bit, integral type in Java. There are no unsigned integral types in Java at all.

Examples of the Unicode Collating Sequence

A < a	U+0041 < U+0061	65 < 97
Z < a	U+005A < U+0061	90 < 97
a < b	U+0061 < U+0062	97 < 98
e < f	U+0063 < U+0064	101 < 102
z < ñ	U+007A < U+00F1	122 < 241
ö < ü	U+00F6 < U+00FC	246 < 252
ı < û	U+0142 < U+0175	322 < 373
ξ < φ	U+03BE < U+03C6	958 < 966
ƒ < ₣	U+222B < U+2247	8747 < 8820
₣ < ⊗	U+2247 < U+2297	8820 < 8895
₣ < ◉	U+2247 < U+2299	8820 < 8897
fi < fl	U+FB01 < U+FB02	64257 < 64258

Lexicographic Ordering

An ordering of characters gives rise to an order on strings of those characters. Strings of characters of (possibly) different lengths are ordered by the first difference in the strings.

Let $<_C$ be the ordering on characters, e.g., $x <_C y$ for any two characters x and y . Let $x_0x_1 \cdots x_{k-1}$ and $y_0y_1 \cdots y_{l-1}$ be two strings of length $k \geq 0$ and $l \geq 0$, respectively. The two strings are equal $x_0x_1 \cdots x_{k-1} = y_0y_1 \cdots y_{l-1}$, if $k = l$ and $x_i = y_i$ for all $0 \leq i \leq k$.

Now let us define lexicographic ordering $<_L$.

Lexicographic ordering. We define $x_0x_1 \cdots x_{k-1} <_L y_0y_1 \cdots y_{l-1}$ if there is an index $i \leq 0$ such that $i < k$ and $i < l$ and $x_i <_C y_i$ and $x_j = y_j$ for all $0 \leq j < i$, or if $l > k$ and for all $0 \leq j < k$ we have $x_j = y_j$.

alligator	<	crocodile
alligator	<	ant
aardvark	<	anteater
ant	<	armadillo
ant	<	anteater
anteater	<	antelope

Notice that the empty strings (sequences of 0 characters) is the smallest string in lexicographic order.

Discrete Math

See, for example, Section 4.3.3 in *Discrete Structures, Logic, and Computability*, 2nd edition, by James L. Hein.

Java Method

A recursive Java method to implement lexicographic ordering on strings based on the Unicode collating sequence:

```
static boolean lexicographic (String x, String y) {  
    if (y.length()==0) return false;  
    else if (x.length()==0) return true;  
    else if (x.charAt(0) < y.charAt(0)) return true;  
    else if (x.charAt(0) > y.charAt(0)) return false;  
    else return lexicographic (  
        x.substring(1), y.substring(1));  
}
```

Dictionary Ordering

Note that for most natural languages lexicographic ordering is not quite “dictionary ordering.” Natural languages often have numerous special rules about: ignorable characters, capitalization, diacritics, digraphs, etc.

Ignorable characters:

dictionary: coal < concentrate < co-operate < corporation

lexicographic: co-operate < coal < concentrate < corporation

Capitalization

dictionary: abduct < Abelian < Aberdeen < abet

lexicographic: Abelian < Aberdeen < abduct < abet

Diacritics

dictionary: cote < côte < coté < côtel

lexicographic: cote < coté < côte < côtel

Digraphs

dictionary: casa < como < chalupa < dónde

lexicographic: casa < chalupa < como < dónde

Other, more complex rules require semantic analysis. Mc=Mac, Mrs.=Mistress, St.=Saint, 1812=Eighteen twelve. Ignoring an initial article, etc. See Knuth, volume 3, pages 8–9.

Time Stamp

Recording the time of an event using a time stamp is a very common task in database and other programs. A good choice for the format of a time stamp is one for which the timestamps when sorted in lexicographic order are also in temporal order.

Aug 8, 2010, 10:56:32.876 am	2010-08-08T10:56:32.876Z
Dec 3, 2010, 9:04:01.327 am	2010-12-03T9:04:01.327Z
Sep 21, 2010, 2:03:11.002 pm	2010-09-21T14:03:11.002Z

Month names when sorted in lexicographic order (even when abbreviated to three characters) are not in chronological order. Also the string of length one "8" is not less than the string of length two "10".

This trick obviates the need for a special timestamp function to compare two timestamps in chronological order. Such a function can be difficult to write correctly due to the irregular nature our society uses in keeping time.

Dictionary Ordering

Java can compare strings in dictionary order using the class

```
java.text.Collator.
```

```
Collator co = Collator.getInstance (Locale.US);  
co.setStrength (Collator.PRIMARY);  
if (co.compare ("abc", "ABC")==0) {  
    System.out.println ("Equivalent.");  
}
```

The difference between “a” and “b” is considered primary, while the difference between “e” and “é” is secondary, and the difference between “e” and “E” is tertiary.

Singleton Pattern

Late creation or frugal management of large objects is often controlled by a static method that creates an instance of a class on behalf of the client.

In this way the correct subclass or implementation can be created.

Also the number of these objects can be controlled so that repeated requests for the object will be fulfilled by returning the same instance.

```
java.util.Calendar.getInstance();
java.text.Collator.getInstance (Locale.US);
java.text.NumberFormat.getInstance (Locale.US);
java.security.KeyFactory.getInstance ("DSA");
java.security.MessageDigest.getInstance ("MD5");
java.awt.AlphaComposite.getInstance (
    AlphaComposite.SR_OUT)
```