Introducing Unicode

Unicode is coded character set (or simply character set) capable of representing most of the writing systems. The recent version of Unicode contains around 138,000 characters covering 150 modern and historic languages and scripts, as well as symbol sets and emoji. The below table shows how some characters from different languages are represent in Unicode.

| **Character** | **Code Point** | **UTF\_8** | **UTF\_16** | **Language** |
| --- | --- | --- | --- | --- |
| a | U+0061 | 61 | 00 61 | English |
| Z | U+005A | 5a | 00 5a | English |
| â | U+00E2 | c3 a2 | 00 e2 | Latin |
| Δ | U+0394 | ce 94 | 03 94 | Latin |
| ع | U+0639 | d8 b9 | 06 39 | Arabic |
| 你 | U+4F60 | e4 bd a0 | 4f 60 | Chinese |
| 好 | U+597D | e5 a5 bd | 59 7d | Chinese |
| ಡ | U+0CA1 | e0 b2 a1 | 0c a1 | Kannada |
| ತ | U+0CA4 | e0 b2 a4 | 0c a4 | Kannada |

Each character or symbol is represented by an unique Code point. Unicode has 1,112,064 code points out of which around 138,000 are presently defined. Unicode code point is represented as U+xxxx where U signifies it as Unicode. The String.codePointAt(int index) method returns code point for character.

String str **=** "你"**;**

**int** codePoint **=** str**.**codePointAt**(**0**);**

System**.**out**.**format**(**"U+%04X"**,** codePoint**);**

*// outputs the code point - U+4F60*

A charset can have one or more encoding schemes and Unicode has multiple encoding schemes such as UTF\_8, UTF\_16, UTF\_16LE and UTF\_16BE that maps code point to bytes.

UTF-8

UTF-8 (8-bit Unicode Transformation Format) is a variable width character encoding capable of encoding all valid Unicode code points using one to four 8-bit bytes. In the above table, we can see that the length of encoded bytes varies from 1 to 3 bytes for UTF-8. Majority of web pages use UTF-8.

The first 128 characters of Unicode, which correspond one-to-one with ASCII, are encoded using a single byte with the same binary value as ASCII. The valid ASCII text is valid UTF-8-encoded Unicode as well.

UTF-16

UTF-16 (16-bit Unicode Transformation Format) is another encoding scheme capable of handling all characters of Unicode character set. The encoding is variable-length, as code points are encoded with one or two 16-bit code units (i.e minimum 2 bytes and maximum 4 bytes).

Many systems such as Windows, Java and JavaScript, internally, uses UTF-16. It is also often used for plain text and for word-processing data files on Windows, but rarely used for files on Unix/Linux or macOS.

Java internally uses UTF-16. From Java 9 onwards, to reduce the memory taken by String objects, it uses either ISO-8859-1/Latin-1 (one byte per character) or UTF-16 (two bytes per character) based upon the contents of the string. [JEPS 254](http://openjdk.java.net/jeps/254).

However don’t confuse the internal charset with Java default charset which is UTF-8. For example, the Strings live in heap memory as UTF-16, however the method String.getBytes() returns bytes encoded as UTF-8, the default charset.

You can use [CharInfo.java](https://github.com/maithilish/codefs/blob/master/misc/char-info/CharInfo.java) to display character details of a string.

To summarize:

* Character set is collection of characters. Numbers, alphabets and Chinese characters are examples of character sets.
* Coded character set is a character set in which each character has an assigned int value. Unicode, US-ASCII and ISO-8859-1 are examples of coded character set.
* Code Point is an integer assigned to a character in a coded character set.
* Character encoding maps between code points of a coded character set and sequences of bytes. One coded character set may have one or more character encodings . For example, ASCII has one encoding scheme while Unicode has multiple encoding schemes - UTF-8, UTF-16, UTF\_16BE, UTF\_16LE etc.

Java IO

Use char stream IO classes Reader and Writer while dealing with text and text files. As already explained, the default charset of Java platform is UTF-8 and text written using Writer class is encoded in UTF-8 and Reader class reads the text in UTF-8.

Using java.io package, we can write and read a text file in default charset as below.

String str **=** "a Z â Δ 你 好 ಡ ತ ع"**;**

File file **=** **new** File**(**"x-utf8.txt"**);**

*// write file in default charset (UTF-8)*

**try** **(**BufferedWriter out **=** **new** BufferedWriter**(new** FileWriter**(**file**)))** **{**

out**.**write**(**str**);**

**}**

*// read file in default charset (UTF-8)*

**try** **(**BufferedReader in **=** **new** BufferedReader**(new** FileReader**(**file**)))** **{**

String line**;**

**while** **((**line **=** in**.**readLine**())** **!=** **null)** **{**

System**.**out**.**println**(**line**);**

**}**

**}**

The above example, uses char stream classes - Writer and Reader - directly that uses default character set (UTF-8).

To encode/decode in non-default charset use byte oriented classes and use a bridge class to convert it char oriented class. For example, to read file as raw bytes use FileInputStream and wrap it with InputStreamReader, a bridge that can encode the bytes to chars in specified charset. Similarly for output, use OutputStreamWriter (bridge) and FileOutputWriter (byte output)

*// read in UTF\_16LE*

InputStream byteInStream **=** **new** FileInputStream**(**file**);**

Reader encodedCharStream **=**

**new** InputStreamReader**(**byteInStream**,** StandardCharsets**.**UTF\_16LE**);**

*// write in UTF\_16LE*

OutputStream byteOutStream **=** **new** FileOutputStream**(**file**);**

Writer decodedCharStream **=**

**new** OutputStreamWriter**(**byteOutStream**,** StandardCharsets**.**UTF\_16LE**);**

Following example, writes a file in UTF\_16BE charset and reads it back.

String str **=** "a Z â Δ 你 好 ಡ ತ ع"**;**

Charset charset **=** StandardCharsets**.**UTF\_16BE**;**

File file **=** **new** File**(**"x-utf16be.txt"**);**

*// write file in non default charset*

**try** **(**BufferedWriter out **=** **new** BufferedWriter**(**

**new** OutputStreamWriter**(new** FileOutputStream**(**file**),** charset**)))** **{**

out**.**write**(**str**);**

**}**

*// read file in non default charset*

**try** **(**BufferedReader in **=** **new** BufferedReader**(**

**new** InputStreamReader**(new** FileInputStream**(**file**),** charset**)))** **{**

String line**;**

**while** **((**line **=** in**.**readLine**())** **!=** **null)** **{**

System**.**out**.**println**(**line**);**

**}**

**}**