Reference


Wikipedia

Java API docs
What is a Character in a String?

<table>
<thead>
<tr>
<th>65</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>B</td>
</tr>
<tr>
<td>67</td>
<td>C</td>
</tr>
</tbody>
</table>

It is a mapping from bits to an element in an alphabet
ASCII

A common mapping

1 - 32 map to control characters
33 - 127 map to characters in the American English alphabet with punctuation

This is an example of a block encoding
   All characters are used by the same number of bits
Variable-length coding (aside)

Use differing amounts of bits to encode characters

Use few bits for frequently occurring characters

Example: Huffman encoding

Example: First MacWrite
ASCII and 128-255

On PCs people started used 128-255 for non-standard characters

Used by non-english languages

Mappings called:
  Code pages
  OEM Code pages

1-127 same as ASCII

Code page 437 - Original IBM code page
Some DOS Code pages

437 — The original IBM PC code page
737 — Greek
775 — Estonian, Lithuanian and Latvian
850 — "Multilingual (Latin-1)" (Western European languages)
852 — "Slavic (Latin-2)" (Central and Eastern European languages)
855 — Cyrillic
857 — Turkish
858 — "Multilingual" with euro symbol
860 — Portuguese
861 — Icelandic
862 — Hebrew
863 — French Canadian
865 — Nordic
866 — Cyrillic
869 — Greek
65001 — UTF-8 Unicode

Yes the code pages were known by number and yes there were multiple code pages for the same language
OEM Pages

OEM pages supported by Windows

Catalog of Character Sets and OEM pages
   http://www.i18nguy.com/unicode/codepages.html
Displaying Characters

Cyrillic alphabet

| а | б | в | г | д | е | ё | ж | з | и | й | к | л | м | н | о | п | р | с | т | у | ф | х | ц | ч | ш | щ |ья | э | ю | я |
| а | б | в | г | д | е | ё | ж | з | и | й | к | л | м | н | о | п | р | с | т | у | ф | х | ц | ч | ш | щ |ья | э | ю | я |

Text mode of VGA-compatible PC graphics hardware uses 8-bit code page

Often done in graphics mode now

Originally used bit maps for characters
Fonts

Description of the visual representation of characters of an "alphabet"

Includes
weight
style
width
serif

The cat in the hat
The cat in the hat
The cat in the hat
The cat in the hat
THE CAT IN THE HAT
THE CAT IN THE HAT
The cat in the hat
THE CAT IN THE HAT
The cat in the hat

THE CAT IN THE HAT

The cat in the hat
Outline Fonts

Adobe Type 1 Fonts
TrueType
OpenType

Rather than use bitmaps for display

Use math functions to describe the outline of each character

Produces smoother characters on screen and print

Can scale the characters to different sizes
So

Code pages map 8 bit words to characters

Also need way to display the characters
Code pages and Standards

ISO/IEC 8859
Standard code pages (8 bit) for latin alphabets
16 parts (pages)
Does not cover East Asian Languages (CJK)
1998
# Unicode

Standard to represent text of any language

## Basic Idea

<table>
<thead>
<tr>
<th>Code Point</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+0041 (65)</td>
<td>A</td>
</tr>
<tr>
<td>U+0042</td>
<td>B</td>
</tr>
</tbody>
</table>

Mapping between numbers and characters
Some Terms

UCS - Universal Character Set
  Standard list of all characters with code point

UTF - Unicode Transformation Format
  Mapping between bits and code points
Some History

1987
  Joe Becker (Xerox), Lee Collins (Apple) and Mark Davis (Apple)
  Start work on a universal character set

1988 - Unicode 88
  Draft calling for 16-bit character model

1991 - Unicode version 1.0.0
  7161 characters

1993 - Unicode 1.1
  34,233 characters
  ISO/IEC 10646-1:1993 (Unicode is also an ISO standard)

2008 - Unicode 5.1
  100,713 characters
  Defines codespace of 1,114,112 code points
Unicode Planes

Unicode code space is divided into planes

Each plane contains 65,535 code points

Plane 0
Basic Multilingual Plane (BMP)
Many symbols (3071)
Contains almost all modern languages
Indic scripts:
- Devanagari, Bengali, Gurmukhi,
- Gujarati, Oriya, Tamil, Telugu,
- Kannada, Malayalam, Sinhala

Plane 1
Supplementary Multilingual Plane (SMP)
Mostly used for historic scripts
Some musical and math symbols

Plane 2 -
Supplementary Ideographic Plane
40,000 seldom seen Han characters

Plane 3-13
Not used
Some Unicode Character Encodings

UTF-8
UTF-16
GB 18030
UTF-8
Variable-length character encoding for Unicode
Uses 1-4 bytes to represent a character

<table>
<thead>
<tr>
<th>Unicode</th>
<th>Byte1</th>
<th>Byte2</th>
<th>Byte3</th>
<th>Byte4</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-127</td>
<td>0xxxxxx</td>
<td></td>
<td></td>
<td></td>
<td>'§' U+0024</td>
</tr>
<tr>
<td>U+000000-U+00007F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 00100100</td>
</tr>
<tr>
<td>0xxxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0x24</td>
</tr>
<tr>
<td>128-2,047</td>
<td>110yyyxx</td>
<td>10xxxxxx</td>
<td></td>
<td></td>
<td>'¢' U+00A2</td>
</tr>
<tr>
<td>U+000080-U+0007FF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 11000010,10100010</td>
</tr>
<tr>
<td>00000yyy xxxxxxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xC2,0xA2</td>
</tr>
<tr>
<td>2,048-65,535</td>
<td>1110yyyy</td>
<td>10yyyyxx</td>
<td>10xxxxxx</td>
<td></td>
<td>'€' U+20AC</td>
</tr>
<tr>
<td>U+000800-U+00FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 11100010,10000010,10101100</td>
</tr>
<tr>
<td>yyyyyyyyy xxxxxxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xE2,0x82,0xAC</td>
</tr>
<tr>
<td>65,536-1,114,111</td>
<td>11110zzz</td>
<td>10zzyyyy</td>
<td>10yyyyxx</td>
<td>10xxxxxx</td>
<td>U+10ABCD</td>
</tr>
<tr>
<td>U+010000-U+00FFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 11110100,10001010,10101111,10001101</td>
</tr>
<tr>
<td>000zzzzz yyyyyyyy xxxxxxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xF4,0x8A,0xAF,0x8D</td>
</tr>
</tbody>
</table>

ASCII encoding is identical to the UTF-8 encoding of same characters

Table is from http://en.wikipedia.org/wiki/UTF-8
Some bit patterns are used to indicate a character needs multi-bytes to encode

So we can only encode 127 character with one byte
UTF-16 & UCS-2

UCS-2
  UCS-2BE (Big Endian)
  UCS-2LE (Little Endian)
    Fix-Length Character encoding
    Uses 2-bytes

UTF-16
  UTF-16 BE (Big Endian)
  UTF-16 LE (Little Endian)
    Variable-Length Character encoding
    Uses 2-bytes words
BOM - Byte Order Mark

Use to indicate Big or Little Endian in UTF-16 and UTC-2

Zero-Width No-Break Space - U+FEFF

FE FF for Big Endian

FF FE for Little Endian

Required for UTC-2

Recommended for UTF-16

Placed before characters
## UTF-16 & UTC-2 Uses

<table>
<thead>
<tr>
<th>UTF-16</th>
<th>UTC-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 2000-Vista</td>
<td>Java before Java 5.0</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>Windows before Windows 2000</td>
</tr>
<tr>
<td>Qualcomm BREW</td>
<td>Symbian OS</td>
</tr>
<tr>
<td>Java</td>
<td>Sony Ericsson UIQ handsets</td>
</tr>
<tr>
<td>.NET</td>
<td>Python</td>
</tr>
</tbody>
</table>

Current version of Python use UTC-4 for plane 1 & 2 characters
How many Encodings are There?

Many more than you would like
## Java Required Encodings

<table>
<thead>
<tr>
<th>Charset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>US-ASCII</td>
<td>Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set</td>
</tr>
<tr>
<td>ISO-8859-1</td>
<td>ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1</td>
</tr>
<tr>
<td>UTF-8</td>
<td>Eight-bit UCS Transformation Format</td>
</tr>
<tr>
<td>UTF-16BE</td>
<td>Sixteen-bit UCS Transformation Format, big-endian byte order</td>
</tr>
<tr>
<td>UTF-16LE</td>
<td>Sixteen-bit UCS Transformation Format, little-endian byte order</td>
</tr>
<tr>
<td>UTF-16</td>
<td>Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark</td>
</tr>
</tbody>
</table>
java.nio.charset.Charset.availableCharsets()
How do we know the Encoding Used

In HTML there is a way to specify it

```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

If it is not specified Web browsers guess

IE does a byte frequency analysis
```
How do we know the Encoding Used

In network protocols you have to specify the encoding
Single Most Important Fact About Encodings

There Ain't No Such Thing As Plain Text.

It does not make sense to have a string without knowing what encoding it uses
Java Strings

Some String Constructors

String(byte[] bytes, String charsetName)
   Constructs a new String by decoding the specified array of bytes using the specified charset.

String(int[] codePoints, int offset, int count)
   Allocates a new String that contains characters from a subarray of the Unicode code point array argument.
String Methods

char charAt(int index)
    Returns the char value at the specified index.

int codePointAt(int index)
    Returns the character (Unicode code point) at the specified index.
Java Streams

Read/Write bytes
Does not know about charsets

Can be used to read Unicode character/other encodings

But your code must convert using correct encoding
Java Reader/Writers

Handle character encodings

But you have to tell it which encoding
InputStreamReader Constructor

InputStreamReader(InputStream in)
   Creates an InputStreamReader that uses the default charset.

InputStreamReader(InputStream in, Charset cs)
   Creates an InputStreamReader that uses the given charset.
Default Encoding in My JRE

java.nio.charset.Charset.defaultCharset()
MacRoman
Don't forget

Networks only deal with bytes

Some protocols are binary so deal only in bytes

Some protocols deal with both binary and text