Module java.base
Package java.nio.charset

Class Charset

java.lang.Object
    java.nio.charset.Charset

All Implemented Interfaces:

`Comparable<Charset>`

```
public abstract class Charset
extends Object
implements Comparable<Charset>
```

A named mapping between sequences of sixteen-bit Unicode code units and sequences of bytes. This class defines methods for creating decoders and encoders and for retrieving the various names associated with a charset. Instances of this class are immutable.

This class also defines static methods for testing whether a particular charset is supported, for locating charset instances by name, and for constructing a map that contains every charset for which support is available in the current Java virtual machine. Support for new charsets can be added via the service-provider interface defined in the `CharsetProvider` class.

All of the methods defined in this class are safe for use by multiple concurrent threads.

Charset names

Charsets are named by strings composed of the following characters:

- The uppercase letters `'A'` through `'Z'` (`'\u0041'` through `'\u005a'`),
- The lowercase letters `'a'` through `'z'` (`'\u0061'` through `'\u007a'`),
- The digits `'0'` through `'9'` (`'\u0030'` through `'\u0039'`),
- The dash character `'-'` (`'\u002d'`, HYPHEN-MINUS),
- The plus character `'+'` (`'\u002b'`, PLUS SIGN),
- The period character `'.'` (`'\u002e'`, FULL STOP),
- The colon character `':'` (`'\u003a'`, COLON), and
- The underscore character `'_'` (`'\u005f'`, LOW LINE).

A charset name must begin with either a letter or a digit. The empty string is not a legal charset name. Charset names are not case-sensitive; that is, case is always ignored when comparing charset names. Charset names generally follow the conventions documented in *RFC 2278: IANA Charset Registration Procedures*.

Every charset has a *canonical name* and may also have one or more *aliases*. The canonical name is returned by the `name` method of this class. Canonical names are, by convention, usually in upper case. The aliases of a charset are returned by the `aliases` method.

Some charsets have an *historical name* that is defined for compatibility with previous versions of the Java platform. A charset's historical name is either its canonical name or one of its aliases. The historical name is returned by the `getEncoding()` methods of the `InputStreamReader` and `OutputStreamWriter` classes.

If a charset listed in the *IANA Charset Registry* is supported by an implementation of the Java platform then its canonical name must be the name listed in the registry. Many charsets are given more than one name in the registry, in which case the registry identifies one of the names as *MIME-preferred*. If a charset has more than one registry name then its canonical name must be the MIME-preferred name and the other names in the registry must be valid aliases. If a supported charset is not listed in the IANA registry then its canonical name must begin with one of the strings `"X-"` or `"x-"`.

The IANA charset registry does change over time, and so the canonical name and the aliases of a particular charset may also change over time. To ensure compatibility it is recommended that no alias ever be removed from a charset, and that if the canonical name of a charset is changed then its previous canonical name be made into an alias.

Standard charsets

Every implementation of the Java platform is required to support the following standard charsets. Consult the release documentation for your implementation to see if any other charsets are supported. The behavior of such optional charsets may differ between implementations.

Charset	Description
`US-ASCII`	Seven-bit ASCII, a.k.a. `ISO646-US`, a.k.a. the Basic Latin block of the Unicode character set
`ISO-8859-1`	ISO Latin Alphabet No. 1, a.k.a. `ISO-LATIN-1`
`UTF-8`	Eight-bit UCS Transformation Format
`UTF-16BE`	Sixteen-bit UCS Transformation Format, big-endian byte order
`UTF-16LE`	Sixteen-bit UCS Transformation Format, little-endian byte order
`UTF-16`	Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark

The `UTF-8` charset is specified by *RFC 2279*; the transformation format upon which it is based is specified in Amendment 2 of ISO 10646-1 and is also described in the *Unicode Standard*.

The UTF-16 charsets are specified by *RFC 2781*; the transformation formats upon which they are based are specified in Amendment 1 of ISO 10646-1 and are also described in the *Unicode Standard*.

The UTF-16 charsets use sixteen-bit quantities and are therefore sensitive to byte order. In these encodings the byte order of a stream may be indicated by an initial *byte-order mark* represented by the Unicode character `'\uFEFF'`. Byte-order marks are handled as follows:

- When decoding, the UTF-16BE and UTF-16LE charsets interpret the initial byte-order marks as a ZERO-WIDTH NON-BREAKING SPACE; when encoding, they do not write byte-order marks.
- When decoding, the UTF-16 charset interprets the byte-order mark at the beginning of the input stream to indicate the byte-order of the stream but defaults to big-endian if there is no byte-order mark; when encoding, it uses big-endian byte order and writes a big-endian byte-order mark.

In any case, byte order marks occurring after the first element of an input sequence are not omitted since the same code is used to represent ZERO-WIDTH NON-BREAKING SPACE.

Every instance of the Java virtual machine has a default charset, which is UTF-8 unless changed in an implementation specific manner. Refer to `defaultCharset()` for more detail.

The `StandardCharsets` class defines constants for each of the standard charsets.

Terminology

The name of this class is taken from the terms used in *RFC 2278*. In that document a *charset* is defined as the combination of one or more coded character sets and a character-encoding scheme. (This definition is confusing; some other software systems define *charset* as a synonym for *coded character set*.)

A *coded character set* is a mapping between a set of abstract characters and a set of integers. US-ASCII, ISO 8859-1, JIS X 0201, and Unicode are examples of coded character sets.

Some standards have defined a *character set* to be simply a set of abstract characters without an associated assigned numbering. An alphabet is an example of such a character set. However, the subtle distinction between *character set* and *coded character set* is rarely used in practice; the former has become a short form for the latter, including in the Java API specification.

A *character-encoding scheme* is a mapping between one or more coded character sets and a set of octet (eight-bit byte) sequences. UTF-8, UTF-16, ISO 2022, and EUC are examples of character-encoding schemes. Encoding schemes are often associated with a particular coded character set; UTF-8, for example, is used only to encode Unicode. Some schemes, however, are associated with multiple coded character sets; EUC, for example, can be used to encode characters in a variety of Asian coded character sets.

When a coded character set is used exclusively with a single character-encoding scheme then the corresponding charset is usually named for the coded character set; otherwise a charset is usually named for the encoding scheme and, possibly, the locale of the coded character sets that it supports. Hence US-ASCII is both the name of a coded character set and of the charset that encodes it, while EUC-JP is the name of the charset that encodes the JIS X 0201, JIS X 0208, and JIS X 0212 coded character sets for the Japanese language.

The native character encoding of the Java programming language is UTF-16. A charset in the Java platform therefore defines a mapping between sequences of sixteen-bit UTF-16 code units (that is, sequences of chars) and sequences of bytes.

Since:

1.4

See Also:

`CharsetDecoder`, `CharsetEncoder`, `CharsetProvider`, `Character`

Constructor Summary

Constructors

Modifier	Constructor	Description
protected	`Charset(String canonicalName, String[] aliases)`	Initializes a new charset with the given canonical name and alias set.

Method Summary

All Methods | Static Methods | Instance Methods | Abstract Methods | Concrete Methods

Modifier and Type	Method	Description
final `Set<String>`	`aliases()`	Returns a set containing this charset's aliases.
static `SortedMap<String, Charset>`	`availableCharsets()`	Constructs a sorted map from canonical charset names to charset objects.
boolean	`canEncode()`	Tells whether or not this charset supports encoding.
final int	`compareTo(Charset that)`	Compares this charset to another.

abstract boolean	**contains**(**Charset** cs)	Tells whether or not this charset contains the given charset.
final **CharBuffer**	**decode**(**ByteBuffer** bb)	Convenience method that decodes bytes in this charset into Unicode characters.
static **Charset**	**defaultCharset**()	Returns the default charset of this Java virtual machine.
String	**displayName**()	Returns this charset's human-readable name for the default locale.
String	**displayName**(**Locale** locale)	Returns this charset's human-readable name for the given locale.
final **ByteBuffer**	**encode**(**String** str)	Convenience method that encodes a string into bytes in this charset.
final **ByteBuffer**	**encode**(**CharBuffer** cb)	Convenience method that encodes Unicode characters into bytes in this charset.
final boolean	**equals**(**Object** ob)	Tells whether or not this object is equal to another.
static **Charset**	**forName**(**String** charsetName)	Returns a charset object for the named charset.
static **Charset**	**forName**(**String** charsetName, **Charset** fallback)	Returns a charset object for the named charset.
final int	**hashCode**()	Computes a hashcode for this charset.
final boolean	**isRegistered**()	Tells whether or not this charset is registered in the IANA Charset Registry.
static boolean	**isSupported**(**String** charsetName)	Tells whether the named charset is supported.
final **String**	**name**()	Returns this charset's canonical name.
abstract **CharsetDecoder**	**newDecoder**()	Constructs a new decoder for this charset.
abstract **CharsetEncoder**	**newEncoder**()	Constructs a new encoder for this charset.
final **String**	**toString**()	Returns a string describing this charset.

Methods declared in class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Details

Charset

```
protected Charset(String canonicalName,
                  String[] aliases)
```

Initializes a new charset with the given canonical name and alias set.

Parameters:

`canonicalName` - The canonical name of this charset

`aliases` - An array of this charset's aliases, or null if it has no aliases

Throws:

`IllegalCharsetNameException` - If the canonical name or any of the aliases are illegal

Method Details

isSupported

```
public static boolean isSupported(String charsetName)
```

Tells whether the named charset is supported.

Parameters:

`charsetName` - The name of the requested charset; may be either a canonical name or an alias

Returns:

`true` if, and only if, support for the named charset is available in the current Java virtual machine

Throws:

`IllegalCharsetNameException` - If the given charset name is illegal

`IllegalArgumentException` - If the given `charsetName` is null

forName

```
public static Charset forName(String charsetName)
```

Returns a charset object for the named charset.

Parameters:

`charsetName` - The name of the requested charset; may be either a canonical name or an alias

Returns:

A charset object for the named charset

Throws:

`IllegalCharsetNameException` - If the given charset name is illegal

`IllegalArgumentException` - If the given `charsetName` is null

`UnsupportedCharsetException` - If no support for the named charset is available in this instance of the Java virtual machine

forName

```
public static Charset forName(String charsetName,
                              Charset fallback)
```

Returns a charset object for the named charset. If the charset object for the named charset is not available or `charsetName` is not a legal charset name, then `fallback` is returned.

Parameters:

`charsetName` - The name of the requested charset; may be either a canonical name or an alias

`fallback` - fallback charset in case the charset object for the named charset is not available or `charsetName` is not a legal charset name. May be `null`

Returns:

A charset object for the named charset, or `fallback` in case the charset object for the named charset is not available or `charsetName` is not a legal charset name

Throws:

`IllegalArgumentException` - If the given `charsetName` is `null`

Since:

18

availableCharsets

```
public static SortedMap<String,Charset> availableCharsets()
```

Constructs a sorted map from canonical charset names to charset objects.

The map returned by this method will have one entry for each charset for which support is available in the current Java virtual machine. If two or more supported charsets have the same canonical name then the resulting map will contain just one of them; which one it will contain is not specified.

The invocation of this method, and the subsequent use of the resulting map, may cause time-consuming disk or network I/O operations to occur. This method is provided for applications that need to enumerate all of the available charsets, for example to allow user charset selection. This method is not used by the `forName` method, which instead employs an efficient incremental lookup algorithm.

This method may return different results at different times if new charset providers are dynamically made available to the current Java virtual machine. In the absence of such changes, the charsets returned by this method are exactly those that can be retrieved via the `forName` method.

Returns:

An immutable, case-insensitive map from canonical charset names to charset objects

defaultCharset

```
public static Charset defaultCharset()
```

Returns the default charset of this Java virtual machine.

The default charset is `UTF-8`, unless changed in an implementation specific manner.

Implementation Note:

An implementation may override the default charset with the system property `file.encoding` on the command line. If the value is `COMPAT`, the default charset is derived from the `native.encoding` system property, which typically depends upon the locale and charset of the underlying operating system.

Returns:

A charset object for the default charset

Since:

1.5

See Also:

file.encoding, native.encoding

name

```
public final String name()
```

Returns this charset's canonical name.

Returns:

The canonical name of this charset

aliases

```
public final Set<String> aliases()
```

Returns a set containing this charset's aliases.

Returns:

An immutable set of this charset's aliases

displayName

```
public String displayName()
```

Returns this charset's human-readable name for the default locale.

The default implementation of this method simply returns this charset's canonical name. Concrete subclasses of this class may override this method in order to provide a localized display name.

Returns:

The display name of this charset in the default locale

isRegistered

```
public final boolean isRegistered()
```

Tells whether or not this charset is registered in the IANA Charset Registry.

Returns:

`true` if, and only if, this charset is known by its implementor to be registered with the IANA

displayName

```
public String displayName(Locale locale)
```

Returns this charset's human-readable name for the given locale.

The default implementation of this method simply returns this charset's canonical name. Concrete subclasses of this class may override this method in order to provide a localized display name.

Parameters:

`locale` - The locale for which the display name is to be retrieved

Returns:

The display name of this charset in the given locale

contains

```
public abstract boolean contains(Charset cs)
```

Tells whether or not this charset contains the given charset.

A charset *C* is said to *contain* a charset *D* if, and only if, every character representable in *D* is also representable in *C*. If this relationship holds then it is guaranteed that every string that can be encoded in *D* can also be encoded in *C* without performing any replacements.

That *C* contains *D* does not imply that each character representable in *C* by a particular byte sequence is represented in *D* by the same byte sequence, although sometimes this is the case.

Every charset contains itself.

This method computes an approximation of the containment relation: If it returns `true` then the given charset is known to be contained by this charset; if it returns `false`, however, then it is not necessarily the case that the given charset is not contained in this charset.

Parameters:

`cs` - The given charset

Returns:

`true` if the given charset is contained in this charset

newDecoder

```
public abstract CharsetDecoder newDecoder()
```

Constructs a new decoder for this charset.

Returns:

A new decoder for this charset

newEncoder

```
public abstract CharsetEncoder newEncoder()
```

Constructs a new encoder for this charset.

Returns:

A new encoder for this charset

Throws:

`UnsupportedOperationException` - If this charset does not support encoding

canEncode

```
public boolean canEncode()
```

Tells whether or not this charset supports encoding.

Nearly all charsets support encoding. The primary exceptions are special-purpose *auto-detect* charsets whose decoders can determine which of several possible encoding schemes is in use by examining the input byte sequence. Such charsets do not support encoding because there is no way to determine which encoding should be used on output. Implementations of such charsets should override this method to return `false`.

Returns:

`true` if, and only if, this charset supports encoding

decode

```
public final CharBuffer decode(ByteBuffer bb)
```

Convenience method that decodes bytes in this charset into Unicode characters.

An invocation of this method upon a charset `cs` returns the same result as the expression

```
cs.newDecoder()
  .onMalformedInput(CodingErrorAction.REPLACE)
  .onUnmappableCharacter(CodingErrorAction.REPLACE)
  .decode(bb);
```

except that it is potentially more efficient because it can cache decoders between successive invocations.

This method always replaces malformed-input and unmappable-character sequences with this charset's default replacement byte array. In order to detect such sequences, use the `CharsetDecoder.decode(java.nio.ByteBuffer)` method directly.

Parameters:

`bb` - The byte buffer to be decoded

Returns:

A char buffer containing the decoded characters

encode

```
public final ByteBuffer encode(CharBuffer cb)
```

Convenience method that encodes Unicode characters into bytes in this charset.

An invocation of this method upon a charset `cs` returns the same result as the expression

```
cs.newEncoder()
  .onMalformedInput(CodingErrorAction.REPLACE)
  .onUnmappableCharacter(CodingErrorAction.REPLACE)
  .encode(bb);
```

except that it is potentially more efficient because it can cache encoders between successive invocations.

This method always replaces malformed-input and unmappable-character sequences with this charset's default replacement string. In order to detect such sequences, use the `CharsetEncoder.encode(java.nio.CharBuffer)` method directly.

Parameters:

`cb` - The char buffer to be encoded

Returns:

A byte buffer containing the encoded characters

encode

```
public final ByteBuffer encode(String str)
```

Convenience method that encodes a string into bytes in this charset.

An invocation of this method upon a charset `cs` returns the same result as the expression

```
cs.encode(CharBuffer.wrap(s));
```

Parameters:

`str` - The string to be encoded

Returns:

A byte buffer containing the encoded characters

compareTo

```
public final int compareTo(Charset that)
```

Compares this charset to another.

Charsets are ordered by their canonical names, without regard to case.

Specified by:

`compareTo` in interface `Comparable<Charset>`

Parameters:

`that` - The charset to which this charset is to be compared

Returns:

A negative integer, zero, or a positive integer as this charset is less than, equal to, or greater than the specified charset

hashCode

```
public final int hashCode()
```

Computes a hashcode for this charset.

Overrides:

`hashCode` in class `Object`

Returns:

An integer hashcode

See Also:

`Object.equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

equals

```
public final boolean equals(Object ob)
```

Tells whether or not this object is equal to another.

Two charsets are equal if, and only if, they have the same canonical names. A charset is never equal to any other type of object.

Overrides:

`equals` in class `Object`

Parameters:

`ob` - the reference object with which to compare.

Returns:

`true` if, and only if, this charset is equal to the given object

See Also:

`Object.hashCode()`, `HashMap`

toString

```
public final String toString()
```

Returns a string describing this charset.

Overrides:

`toString` in class `Object`

Returns:

A string describing this charset