Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Input Output in Java : Char based/5b766bef4227cf4f24a3066a

TUTORIAL

Input Output in Java : Char based

Topics

Java internally stores characters (`char` type) in 16-bit UCS-2 character set. But the external data source/sink could store characters in other character set (e.g., US-ASCII, ISO-8859-x, UTF-8, UTF-16, and many others), in fixed length of 8-bit or 16-bit, or in variable length of 1 to 4 bytes. The byte/character streams refer to the unit of operation within the Java programs, which does not necessary correspond to the amount of data transferred from/to the external I/O devices. This is because some charsets use fixed-length of 8-bit (e.g., US-ASCII, ISO-8859-1) or 16-bit (e.g., UCS-16), while some use variable-length of 1-4 bytes (e.g., UTF-8, UTF-16, UTF-16-BE, UTF-16-LE, GBK, BIG5). When a character stream is used to read an 8-bit ASCII file, an 8-bit data is read from the file and put into the 16-bit char location of the Java program. Java provides following classes for character based input and output: -

```
Reader
                InputStreamReader
                                FileReader
                BufferedReader
                                LineNumberReader
                FilterReader
                                PushbackReader
                CharArrayReader
                PipedReader
                StringReader

Writer
                OutputStreamWriter
                                FileWriter
                BufferedWriter
                FilterWriter
                CharArrayWriter
                PipedWriter
                StringWriter
```

Abstract superclass Reader and Writer

Other than the unit of operation and charset conversion (which is extremely complex), character-based I/O is almost identical to byte-based I/O. Instead of InputStream and OutputStream, we use Reader and Writer for character-based I/O. The abstract superclass Reader operates on char. It declares an abstract method read() to read one character from the input source. read() returns the character as an int between 0 to 65535 (a char in Java can be treated as an unsigned 16-bit integer); or -1 if end-of-stream is detected; or throws an IOException if I/O error occurs. There are also two variations of read() to read a block of characters into char-array.

```
public abstract int read() throws IOException
public int read(char[] chars, int offset, int length) throws IOException
public int read(char[] chars) throws IOException
```

The abstract superclass Writer declares an abstract method write(), which writes a character to the output sink. The lower 2 bytes of the int argument is written out; while the upper 2 bytes are discarded.

```
public void abstract void write(int aChar) throws IOException
public void write(char[] chars, int offset, int length) throws IOException
public void write(char[] chars) throws IOException
```

File I/O Character-Streams - FileReader & FileWriter

FileReader and FileWriter are concrete implementations to the abstract superclasses Reader and Writer, to support I/O from disk files. FileReader/FileWriter assumes that the default character encoding (charset) is used for the disk file. The default charset is kept in the JVM's system property "file.encoding". You can get the default charset via static method java.nio.charset.Charset.defaultCharset() or System.getProperty("file.encoding"). It is probable safe to use FileReader/FileWriter for ASCII texts, provided that the default charset is compatible to ASCII (such as US-ASCII, ISO-8859-x, UTF-8, and many others, but NOT UTF-16, UTF-16BE, UTF-16LE and many others). Use of FileReader/FileWriter is NOT recommended as you have no control of the file encoding charset.

Buffered I/O Character-Streams - BufferedReader & BufferedWriter

BufferedReader and BufferedWriter can be stacked on top of FileReader/FileWriter or other character streams to perform buffered I/O, instead of character-by-character. BufferedReader provides a new method readLine(), which reads a line and returns a String (without the line delimiter). Lines could be delimited by "\n" (Unix), "\r\n" (Windows), or "\r" (Mac).

Following example will show the writing and reading to a file using BufferedWriter and BufferedReader: -

Java
```
import java.io.*;

```

```java
class Main
{
  public static void main(String[] args)
  {
    String strFilename = "cq1.txt";
    String message = "Code Quotient\nGet better at Coding.\n";

    // Print the default charset
    System.out.println("Charset = " +
java.nio.charset.Charset.defaultCharset());

    try (BufferedWriter out = new BufferedWriter(new
FileWriter(strFilename)))
    {
      out.write(message);
      out.flush();
    } catch (IOException ex) {
      ex.printStackTrace();
    }

    try (BufferedReader in = new BufferedReader(new
FileReader(strFilename)))
    {
      String inLine;
      while ((inLine = in.readLine()) != null) {
        System.out.println(inLine);
      }
    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

Text File I/O - InputStreamReader and OutputStreamWriter

As mentioned, Java internally stores characters (char type) in 16-bit UCS-2 character set. But the external data source/sink could store characters in other character set (e.g., US-ASCII, ISO-8859-x, UTF-8, UTF-16, and many others), in fixed length of 8-bit or 16-bit, or in variable length of 1 to 4 bytes. The FileReader/FileWriter introduced earlier uses the default charset for decoding/encoding, resulted in non-portable programs.

To choose the charset, you need to use InputStreamReader and OutputStreamWriter. InputStreamReader and OutputStreamWriter are considered to be byte-to-character "bridge" streams. You can choose the character set in the InputStreamReader's constructor:

```java
public InputStreamReader(InputStream in)   // Use default charset
public InputStreamReader(InputStream in, String charsetName) throws
```

```
UnsupportedEncodingException
public InputStreamReader(InputStream in, Charset cs)
```

You can list the available charsets via static method java.nio.charset.Charset.availableCharsets(). The commonly-used Charset names supported by Java are:

- **US-ASCII:** 7-bit ASCII (aka ISO646-US)
- **ISO-8859-1:** Latin-1
- **UTF-8:** Most commonly-used encoding scheme for Unicode
- **UTF-16BE:** Big-endian (big byte first) (big-endian is usually the default)
- **UTF-16LE:** Little-endian (little byte first)
- **UTF-16:** with a 2-byte BOM (Byte-Order-Mark) to specify the byte order. FE FF indicates big-endian, FF FE indicates little-endian.

As the InputStreamReader/OutputStreamWriter often needs to read/write in multiple bytes, it is best to wrap it with a BufferedReader/BufferedWriter.

PrintStream & PrintWriter

The byte-based java.io.printSteam supports convenient printing methods such as print() and println() for printing primitives and text string. Primitives are converted to their string representation for printing. The printf() and format() were introduced in JDK 1.5 for formatting output with former specifiers. printf() and format() are identical.

A PrintStream never throws an IOException. Instead, it sets an internal flag which can be checked via the checkError() method. A PrintStream can also be created to flush the output automatically. That is, the flush() method is automatically invoked after a byte array is written, one of the println() methods is invoked, or after a newline ('\n') is written.

The standard output and error streams (System.out and System.err) belong to PrintStream.

All characters printed by a PrintStream are converted into bytes using the default character encoding. The PrintWriter class should be used in situations that require writing characters rather than bytes.

The character-stream PrintWriter is similar to PrintStream, except that it write in characters instead of bytes. The PrintWriter also supports all the convenient printing methods print(), println(), printf() and format(). It never throws an IOException and can optionally be created to support automatic flushing.