

## Chapter 16 Simple Input and Output

1. The `\` is a special character. It should be written as `\\` in Java using the Escape sequence.
2. Use `exists()` in the `File` class to check whether a file exists. Use `delete()` in the `File` class to delete this file. Use `renameTo(File)` to rename the name for this file. You cannot find the file size using the `File` class.
3. Use `File.separator` and `File.separatorChar` to obtain file and directory separator string and character. Use `File.pathSeparator` and `File.pathSeparatorChar` to obtain path separator string and character.
4. No. The `File` class can be used to obtain file properties and manipulate files, but cannot perform I/O.
5. You have to use Java I/O classes to create objects and use the methods in the objects to perform I/O. A Java I/O object is called a stream. An object for reading data is called an input stream and an object for writing data is called an output stream.
6. Although it is not technically precise, a text file consists of a sequence of characters and a binary file consists of a sequence of bits. You can use a text editor to view a text file, but not a binary file.
7. Characters are represented using Unicode in the memory and characters are represented in a file using a specified encoding scheme. If no encoding scheme is specified, the system's default encoding scheme is used.
8. `read()` returns the Unicode for the character. The exact value stored in the file is not necessary the Unicode. Java converts the value from the file to a Unicode based on the encoding used in the file.
9. No. The Unicode value is automatically converted into an appropriate value based on the file-encoding scheme.
10. Use `new FileReader(File)` or `new FileReader(filename)` to create an input stream from a text file. If a file does not exist, a `FileNotFoundException` would occur. Use `new FileWriter(File)` or `new FileWriter(filename)` to create an output stream to write to a file. If a file already exists, a new file will be created. To append data to an existing file, use `new FileWriter(File, true)` or `new FileWriter(filename, true)` to create a `FileWriter`.

11. All the methods and constructors in Java I/O classes except `PrintWriter` and `PrintStream` throw `IOException`. Because there are always some unexpected situation may arise.
12. For a string "91", the characters '9' and '1' are written to a file using `FileWriter`. Characters are encoded into bytes according to the encoding scheme used for the file.
13. Two reasons: (1) closing a stream ensures that data will be written to the file. (2) closing a stream release resource acquired by the stream object.
14. The line separator is not necessarily '\n'. To obtain a system-specific line separator, use

```
static String lineSeparator = (String)java.security.  
AccessController.doPrivileged(  
new sun.security.action.GetPropertyAction("line.separator"));
```

15. Since physical input and output involving I/O devices are typically very slow compared with CPU processing speeds, you should use buffered input/output streams to improve performance. You can create a buffer stream for input from any instance of `Reader` and create a buffer stream for output from any instance of `Writer`.

Both of the following statements are correct.

```
PrintWriter output1 = new PrintWriter(  
new BufferedWriter(new FileWriter("out1.txt")));
```

```
BufferedWriter output2 = new BufferedWriter(  
new PrintWriter(new FileWriter("out2.txt")));
```

But the second is better replaced by

```
BufferedWriter output2 = new BufferedWriter(  
new FileWriter("out2.txt"));
```

because it has no benefits to add a print stream between a `FileWriter` and `BufferedWriter`.

16. Use `PrintWriter` and `PrintStream` to write primitive values, strings, and object's `toString()` value as text. `PrintStream` is used for console output. In all other cases, you should use `PrintWriter`. To create an instance of `PrintWriter`, use `new PrintWriter(filename)` or `PrintWriter(File)`. The type for `System.in`, `System.out`, and `System.err` is `PrintStream`. `PrintWriter` and `PrintStream` don't throw `IOException`. You don't have to place them in a try-catch block.
17. In `PrintWriter`, you can use print statement. In `BufferedWriter`, you have to convert a primitive type value to a string, then write the string.
18. Binary I/O reads a byte from a file and copies it directly to the memory with any conversion, vice versa. Text I/O requires encoding and decoding. The JVM

converts a Unicode to a file specific encoding when writing a character and converts a file specific encoding to a Unicode when reading a character.

19. The value of a byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned. The only abstract method in `InputStream` is `read()` and the only abstract method in `OutputStream` is `write(int)`.
20. All the methods in `FileInputStream/FileOutputStream` are inherited from `InputStream/OutputStream`. Use `new FileInputStream(filename)` or `new FileInputStream(File)` to create a new `FileInputStream` and use `new FileOutputStream(filename)`, `new FileOutputStream(File)`, `new FileOutputStream(filename, true)` or `new FileOutputStream(File, true)` to create a `FileOutputStream`.
21. A `FileNotFoundException` would occur if you attempt to create an input stream for a nonexistent file. You can append data in an existent file if the output stream is created using `new FileOutputStream(filename, true)` or `new FileOutputStream(File, true)`. Otherwise, the file is overridden if it already exists.
22. Invoking `read()` reads one byte from the input. So, `input.available()` returns 99. After invoking `readInt()`, `input.available()` returns  $99 - 4 = 95$ . After invoking `readChar()`, `input.available()` returns  $95 - 2 = 93$ . After invoking `readDouble()`, `input.available()` returns  $93 - 8 = 85$ .
23. `writeByte(91)` writes one byte for number 91 (0x5B in hex, 01011011 in binary) is written to a file using `FileOutputStream`.
24. The `available()` method returns the available bytes in the stream. `available() == 0` indicates the end of a file.
25. Since `java.io.FileNotFoundException` is a subclass of `IOException`, the catch clause for `java.io.FileNotFoundException` should be put before the catch clause for `java.io.IOException`.
26. Java uses Unicode, but Windows uses ASCII. The Unicode is converted to ASCII code when writing a character. After the program is finished, the file will contain eight bytes, each represents an ASCII code. So, the values are  
31 32 33 34 35 36 37 38

Note the ASCII code in hex for character 1 is 31.

27. Each int value takes four bytes. Since two int values are written into the file, the file contains eight bytes. The values are  
00 00 04 D2 00 00 16 2E

The first four bytes are for 1234, which equals to 4D2 in hex, and the second byte is for 5678, which equals to 2246 in hex.

28.

```
output.writeChar('A'); => 2 bytes  
output.writeChars("BC"); => 4 bytes  
output.writeUTF("DEF"); => 2 + 3 bytes (the first two bytes store the number  
of characters in the string. Each ASCII character takes one byte in UTF)
```

29. Since physical input and output involving I/O devices are typically very slow compared with CPU processing speeds, buffered input/output streams can be used to improve performance. You can create a buffered input stream by wrapping a `BufferedInputStream/BufferedReader` on any instance of `InputStream/Reader`, and create a buffered output stream by wrapping a `BufferedOutputStream/BufferedWriter` on any instance of `BufferedOutputStream/Writer`.

30. Any objects that are instance of `Serializable` may be stored using the object stream. You use the `writeObject` method to write an object to the object output stream and use `readObject` to read an object from the object input stream. The `readObject` method returns a value of the `Object` type.

31. No. An object may not be serialized even though its class implements `java.io.Serializable`, because it may contain non-serializable instance variables. Implementing `java.io.Serializable` is a necessary requirement for serialization, but not sufficient. You still have to ensure that all the variables in the object are serializable. A static variable is not serialized. If you don't want a variable to be serialized, mark it `transient`.
32. Yes. An array can be serialized if all its elements can be serialized.
33. Yes. Because `ObjectInputStream/ObjectOutputStream` contains all features and operations in `DataInputStream/DataOutputStream`.
34. A `java.io.NotSerializableException` would occur.

35. Yes, because they share the same interface for reading and writing data in the same format. No. Cannot write objects.

36. `RandomAccessFile raf = new RandomAccessFile("address.dat", "rw");`

```
DataOutputStream outfile = new DataOutputStream(new  
FileWriter("address.dat"));
```

To create a `RandomAccessFile` stream, you simply use the `RandomAccessFile` constructor. To create a `DataOutputStream`, you use `DataOutputStream` wrapped on `FileOutputStream`.

37. It will compile fine, but raises a run time exception on invoking `readInt()` because nothing is in the file.

