

## Chapter 18 Binary I/O

1. 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.
2. 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.
3. 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.
4. 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.
5. The values stored in the text file are 0x41 0x42 0x43.
6. If you write string "100" to an ASCII text file, the values stored are 0x31 0x30 0x30. If you write a numeric byte-type value 100 using binary I/O, the value stored in the file is 0x64.
7. The encoding scheme for representing a character in a Java program is the Unicode. By default, a text file is encoded using ASCII.
8. Almost all the methods and constructors in Java I/O classes, because there are always some unexpected situation may arise during I/O.
9. 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.
10. 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)`.
11. 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.

12. 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.
13. An instance of FileInputStream can be used as an argument to construct a Scanner and an instance of FileOutputStream can be used as an argument to construct a Formatter. So you can create a Formatter to append text into a file using

```
new Formatter(new FileOutputStream("temp.txt", true));
```

If temp.txt does not exist, it is created. If temp.txt already exists, new data is appended to the file.

14. 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  $93 - 8 = 85$ .
15. writeByte(91) writes one byte for number 91 (0x5B in hex, 01011011 in binary) is written to a file using FileOutputStream.
16. The available() method returns the available bytes in the stream. available() == 0 indicates the end of a file.
17. 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.
18. 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.
19. 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.

20.

```
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)
```

21. 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`.

22. 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.

23 No. For example, two `ArrayList` objects may have different size, so they are serialized into different sizes. Consider two `JButton` objects. If one button has an icon, and the other does not, the one with an icon will require more space in the file when it is serialized.

24 o. 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`.

25 Yes. An array can be serialized if all its elements can be serialized.

26 Yes. Because `ObjectInputStream/ObjectOutputStream` contains all features and operations in `DataInputStream/DataOutputStream`.

27 A `java.io.NotSerializableException` would occur.

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

29. `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`.

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