# CS209A - File IO

## Key Content

- File and Path
- I/O Stream
- Charsets
- Some pitfalls

### 1. Class `File` basic usage

1.1 File and path

The file path should be passed into the constructor of `File` when new a `File` object.

```
File f = new File("/Users/zhaoyao/Downloads/1.jpeg");
```

When constructing a `File` object, either an absolute path or a relative path can be passed as input parameter. The absolute path is the full path beginning with the root directory, for example:

```
File f = new File("c:\\windows\\1.jpeg");
```

Note that the Windows platform uses '\'as a path separator, while Linux platform uses '/' as a path separator.

**Relative path**

When passing in a relative path, the current directory appends the relative path forms an absolute path:

```
// Assume current directory is c:\windows
File f1 = new File("1.jpeg"); // absolute path: "c:\\windows\\1.jpeg"
File f2 = new File(".\\1.jpeg");// absolute path: "c:\\windows\\1.jpeg"
File f3 = new File("..\\1.jpeg");// absolute path: "c:\\1.jpeg"
```

Note: "."represent the current directory and ".."the parent directory.

**Canonical path**

What is canonical path? What is the difference between absolute path and canonical path?

Please run the following code and observe the result :

```
File f = new File("..");
System.out.println(f.getPath());
System.out.println(f.getAbsolutePath());
System.out.println(f.getCanonicalPath());
```

**An absolute path maybe contains "." And "..", canonical path is a standard absolute path which will not contain "." And "..".**

1.2 File and Directory

`File` Objects can represent files or directories. When a new `File` Object is created successfully, it doesn't mean the file or directory exists, because creating a file object doesn't cause any disk operations.
When does the disk operation actually performed ?
The methods of the `File` Object are invoked.
Please run the following code and observe the result :

```
File f1 = new File("C: \\Windows");
File f2 = new File("C: \\Windows\\notepad.exe");
File f3 = new File("C: \\Windows\\null");
System.out.println(f1.isFile());
System.out.println(f1.isDirectory());
System.out.println(f2.isFile());
System.out.println(f2.isDirectory());
System.out.println(f3.isFile());
System.out.println(f3.isDirectory());
```

You can use the following methods to get more information of the file:

- boolean canRead()
- boolean canWrite()
- boolean canExecute()
- long length()

Create and delete files:

- boolean createNewFile()
- boolean delete()

Create and delete directories:

- boolean mkdir():Create the directory represented by the current File object;
- boolean mkdirs():Create the directory represented by the current File object, and if necessary, create the non-existent parent directory;
- boolean delete():Delete the directory represented by the current File object. The current directory must be empty, otherwise it can't be deleted successfully.

Traverse files and directories:

- File[] listFiles(): if the current object is a directory, the method can list all the names of files and subdirectories under the directory

Please run the following code and observe the result :

```
File f = new File("C: \\Windows");
File[] fs = f.listFiles();
if (fs != null) {
    for (File f : files) {
        System.out.println(f);
    }
}
```

More detail: https://docs.oracle.com/javase/tutorial/essential/io/fileio.html

### 1.3 Class `Path` basic usage

JDK also provides class `Path` to do file operations. The usage of class `Path` is similar with the class `File`, and even simper.

A `Path` instance represents a path in the file system. A path can point to either a file or a directory.

```
Path p1 = Paths.get(".", "project", "study"); // create a Path object
System.out.println(p1);
Path p2 = p1.toAbsolutePath(); // convert to a canonical path
System.out.println(p2);
Path p3 = p2.normalize(); // convert to a canonical path
System.out.println(p3);
File f = p3.toFile(); // convert to a File object
System.out.println(f);
for (Path p : Paths.get("..").toAbsolutePath()) { // trace back the Path
    System.out.println("    " + p);
}
```

### 1.4 Class `Files` basic usage

The `java.nio.file.Files` class works with `java.nio.file.Path` instances, so you need to understand the `Path` class before you can work with the `Files` class.

Files.exists()method checks if a given Path exists in the file system.

```
Path path = Paths.get("C: \\Windows\\null ");
boolean pathExists = Files.exists(path);
```

The Files.createDirectory() method creates a new directory from a Path instance.

```
Path path = Paths.get("C: \\Windows\\null ");
try {
    Path newDir = Files.createDirectory(path);
} catch(FileAlreadyExistsException e){
    System.out.println("The directory already exists." );
} catch (IOException e) {
    //something else went wrong
    e.printStackTrace();
```

```
}
```

The Files.copy() method copies a file from one path to another.

```
Path sourcePath      = Paths.get("C: \\Windows\\1.txt ");
Path destinationPath = Paths.get("C: \\Windows\\documents\\1_copy.txt ");

try {
    Files.copy(sourcePath, destinationPath);
} catch(FileAlreadyExistsException e) {
    System.out.println( "Destination file already exists. " );
} catch (IOException e) {
    //something else went wrong
    e.printStackTrace();
}
```

It is possible to force the Files.copy() to overwrite an existing file, append the copy option when copying.

```
Files.copy(sourcePath, destinationPath,
                StandardCopyOption.REPLACE_EXISTING);
```

The Files.move() method moves a file from one path to another and can change its name in the same operation

```
Path sourcePath      = Paths.get("C: \\Windows\\1.txt ");
Path destinationPath = Paths.get("C: \\Windows\\documents\\2_move.txt ");
try {
    Files.move(sourcePath, destinationPath,
                StandardCopyOption.REPLACE_EXISTING);
} catch (IOException e) {
    //moving file failed.
    e.printStackTrace();
```

```
}
```

First the source path and destination path are created. The source path points to the file to move, and the destination path points to where the file should be moved to. Then the Files.move() method is called. This results in the file being moved.

Notice the third parameter passed to Files.move() . This parameter tells the Files.move() method to overwrite any existing file at the destination path. This parameter is actually optional.

The Files.move() method may throw an IOException if moving the file fails. For instance, if a file already exists at the destination path, and you have left out the StandardCopyOption.REPLACE_EXISTING option, or if the file to move does not exist etc.

The Files.delete() method can delete a file or directory.

```
Path path = Paths.get("C: \\Windows\\1.txt ");
try {
    Files.delete(path);
} catch (IOException e) {
    //deleting file failed
    e.printStackTrace();
}
```

The java.nio.file.Files class contains many other useful functions, like functions for creating symbolic links, determining the file size, setting file permissions etc. Check out the JavaDoc for the java.nio.file.Files class for more information about these methods.

1.5 Sample code

```java
public static boolean createFile(String destFileName) {
    File file = new File(destFileName);
    if (file.exists()) {
        System.out.println("Create single file " + destFileName + " fail,target file already exists! ");
        return false;
    }
    if (destFileName.endsWith(File.separator)) {
        System.out.println("Create single file " + destFileName + " fail, target file cannot be a directory! ");
        return false;
    }
    // Check if the directory where the target file is located exists
    if (!file.getParentFile().exists()) {
        // if the directory where the target file is located doesn't exist, create
        // its' parent directory.
        System.out.println("directory where target file is located doesn't exist, create its' parent directory! ");
        File parentFile = file.getParentFile();
        parentFile.mkdirs();
        if (!file.getParentFile().mkdirs()) {
            System.out.println("Create directory where target file is located fails! ");
            return false;
        }
    }
    // Create target file
    try {
        if (file.createNewFile()) {
            System.out.println("Create single file " + destFileName + " success! ");
            return true;
        } else {
            System.out.println("Create single file " + destFileName + " fail! ");
            return false;
        }
    } catch (IOException e) {
        e.printStackTrace();
        System.out.println("Create single file " + destFileName + " fail! " + e.getMessage());
        return false;
    }
```

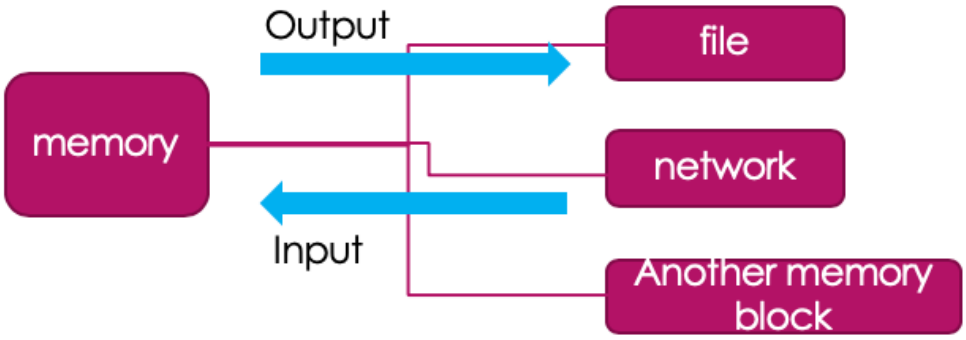
Invoke above method and observe result.

How to do the same operations using `Path` and `Files`?

## 2. I/O streams

### 2.1 I/O streams

A computer can be connected to many different types of input and output devices. If a programming language had to deal with each type of device as a special case, the complexity would be overwhelming. One of the major achievements in the history of programming has been to come up with good abstractions for representing I/O devices. In Java, the main I/O abstractions are called I/O streams.




Files are common sources and destination for an IO stream.

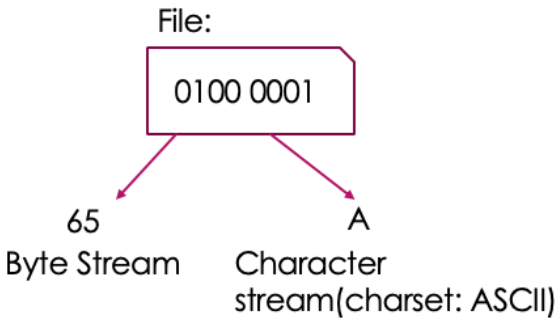
2.2 Byte and Character Streams

**Byte streams:**

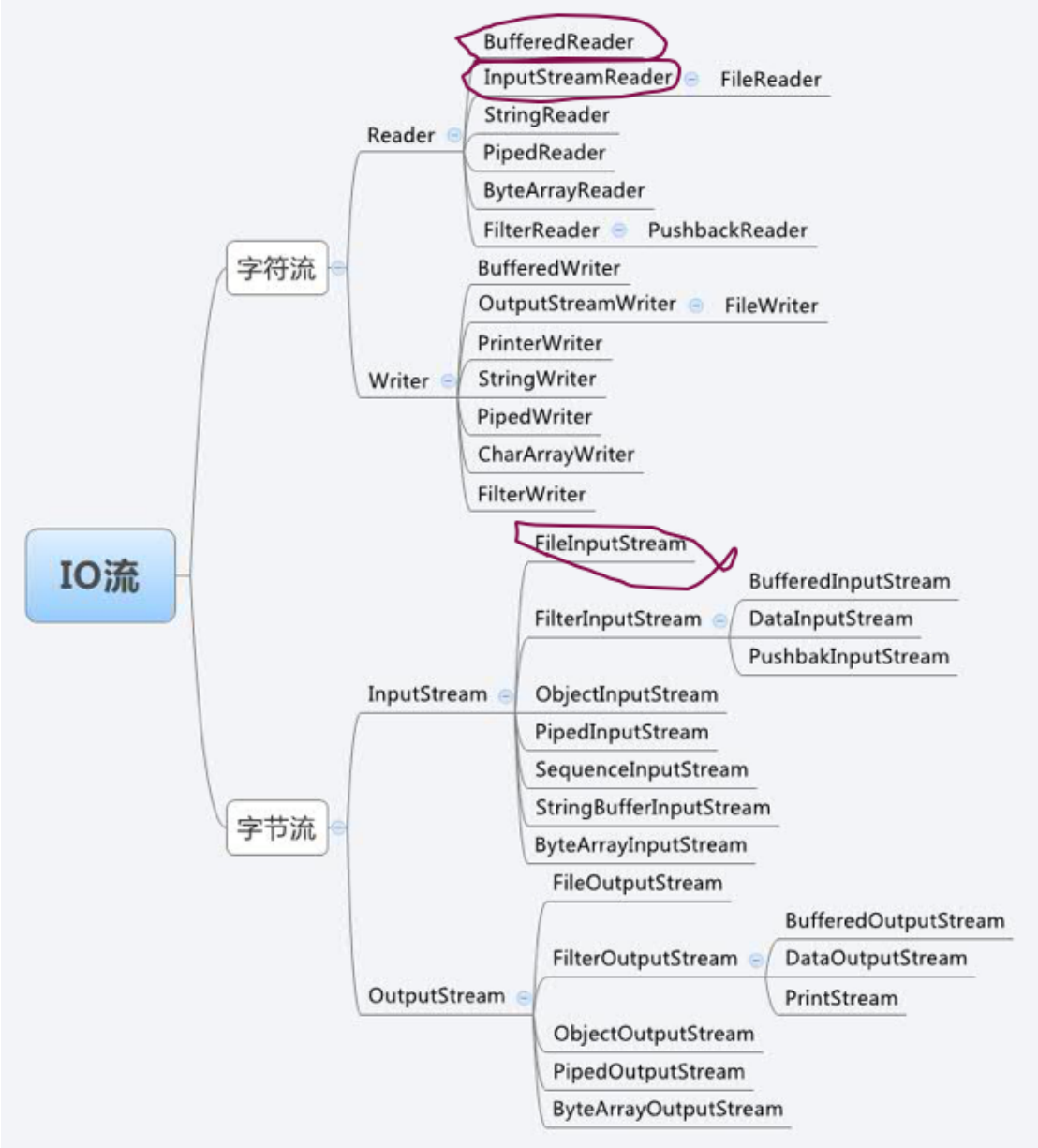
A byte stream is for machine-formatted data, is represented in binary form, the same way that data is represented inside the computer, that is, as strings of zeros and ones.

**Character streams:**

A character stream is for human-readable data – for instance text in English or Chinese. To work, the essence is to look up the specified charset (such as utf-8,utf-16) when reading based on the byte stream. Because when working with text data, the same code can represent characters with different ways (please see the prac intro video).




2.3 JAVA IO Stream Class Structure




Character streams are often "wrappers" for byte streams. The character stream uses the byte stream to perform the physical I/O, while the character stream handles translation between characters and bytes. `FileReader`, for example, uses `FileInputStream`, while `FileWriter` uses `FileOutputStream`.

There are two general-purpose byte-to-character "bridge" streams: `InputStreamReader` and `OutputStreamWriter`. Use them to create character streams when there are no prepackaged character stream classes that meet your needs.

## 2.4 Sample Code

### 2.4.1 FileInputStream

`FileInputStream` obtains input bytes from a file in a file system.

**Parent class**： `InputStream`

**Other related classes**： `ByteArrayInputStream`, `StringBufferInputStream`, and `FileInputStream` are three basic media streams that read data from Byte arrays, stringbuffers, and local files, respectively. The `PipedInputStream` reads data from a pipe, often a pipe can be used to provide shared memory among several threads.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;


public class ByteReader {

    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("sample.txt")){

            byte[] buffer = new byte[65535];

            int byteNum = fis.read(buffer);
            for(int i = 0; i < byteNum; i++){
                System.out.printf("%02x ",buffer[i]);
            }
            System.out.println();

        } catch (FileNotFoundException e) {
            System.out.println("The pathname does not exist.");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Failed or interrupted when doing the I/O operations");
            e.printStackTrace();
        }

    }

}
```

提升io的效率

Observe the result.

2.4.2InputStreamReader

`InputStreamReader` is a bridge between a byte stream and a character stream that converts a byte stream into a character stream.

```java
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;

public class StreamReader {

    public static void main(String[] args) {

        try (InputStreamReader isr = new InputStreamReader(new FileInputStream("sample.txt"), "gb18030")) {

            char[] cbuf = new char[65535];
            int file_len = isr.read(cbuf);

            System.out.println(file_len);
            System.out.println(cbuf);

        } catch (FileNotFoundException e) {
            System.out.println("The pathname does not exist.");
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            System.out.println("The Character Encoding is not supported.");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Failed or interrupted when doing the I/O operations");
            e.printStackTrace();
        }
    }
}
```

字符集编码

Observe the result.

2.4.3BufferedReader

If have no buffer, each read or write request is handled directly by the underlying OS. This can make a program much less efficient, since each such request often triggers disk access, network activity, or some other operation that is relatively expensive.

To reduce this kind of overhead, the Java platform implements buffered I/O streams. Buffered input streams read data from a memory area known as a buffer; the native input API is called only when the buffer is empty. Similarly, buffered output streams write data to a buffer, and the native output API is called only when the buffer is full.

There are four buffered stream classes used to wrap unbuffered streams: `BufferedInputStream` and `BufferedOutputStream` create buffered byte streams, while `BufferedReader` and `BufferedWriter` create buffered character streams.

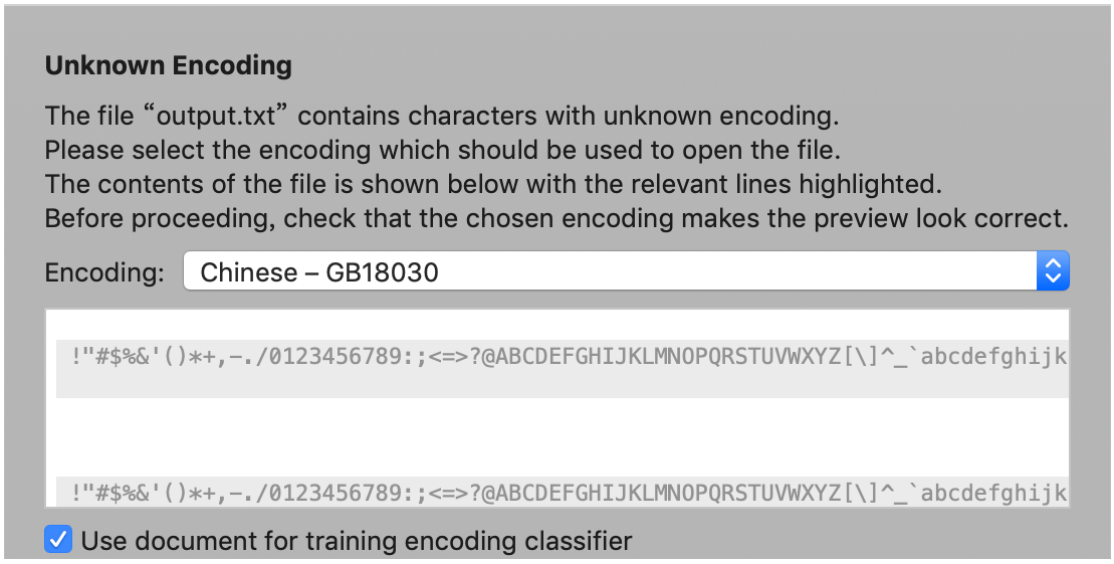
```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;

public class BufferReader {

    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream(new File("sample.txt"));
                InputStreamReader isr = new InputStreamReader(fis, "gb18030");
                BufferedReader bReader = new BufferedReader(isr);){


            char[] cbuf = new char[65535];
            int file_len = bReader.read(cbuf);

            System.out.println(file_len);
            System.out.println(cbuf);

        } catch (FileNotFoundException e) {
            System.out.println("The pathname does not exist.");
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            System.out.println("The Character Encoding is not supported.");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Failed or interrupted when doing the I/O operations");
            e.printStackTrace();
        }
    }
}
```

Observe the result.

2.4.4FileOutputStream

```java
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class ByteWriter {

    public static void main(String[] args) {
        try (FileOutputStream fos = new FileOutputStream("output.txt")){

            byte[] buffer = new byte[65535];
            for(int i = 0; i < buffer.length; i++){
                buffer[i] = (byte) i;
            }

            fos.write(buffer);
            fos.flush();//fos.close();
        } catch (FileNotFoundException e) {
            System.out.println("The pathname does not exist.");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Failed or interrupted when doing the I/O operations");
            e.printStackTrace();
        }

    }

}
```

When you try to open the output.txt, it is possible that you will encounter a problem like this:




TO solve this problem should open a binary document with Notepad++(install HexEditor) / VS Code(install extension:Hexdump)/Sublime Text(install plugin: HexViewer) /UltraEdit and so on.

```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   ................
00000010: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F   ................
00000020: 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   .!"#$%&'()*+,-./
00000030: 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F   0123456789:;<=>?
00000040: 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F   @ABCDEFGHIJKLMNO
00000050: 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F   PQRSTUVWXYZ[\]^_
00000060: 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F   `abcdefghijklmno
00000070: 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F   pqrstuvwxyz{|}~.
00000080: 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F   ................
00000090: 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F   ................
000000a0: A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF   .!"#$%&'()*+,-./
000000b0: B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF   0123456789:;<=>?
000000c0: C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF   @ABCDEFGHIJKLMNO
000000d0: D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF   PQRSTUVWXYZ[\]^_
000000e0: E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF   `abcdefghijklmno
000000f0: F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF   pqrstuvwxyz{|}~.
00000100: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   ................
00000110: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F   ................
00000120: 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   .!"#$%&'()*+,-./
00000130: 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F   0123456789:;<=>?
00000140: 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F   @ABCDEFGHIJKLMNO
00000150: 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F   PQRSTUVWXYZ[\]^_
00000160: 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F   `abcdefghijklmno
00000170: 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F   pqrstuvwxyz{|}~.
00000180: 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F   ................
00000190: 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F   ................
000001a0: A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF   .!"#$%&'()*+,-./
000001b0: B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF   0123456789:;<=>?
000001c0: C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF   @ABCDEFGHIJKLMNO
000001d0: D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF   PQRSTUVWXYZ[\]^_
000001e0: E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF   `abcdefghijklmno
000001f0: F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF   pqrstuvwxyz{|}~.
00000200: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   ................
```

If you open it with UTF-8:

```
1
2
3
4  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
5
6
7  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
8
9
10 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
11
12
13 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

If you open it with UTF-16 BE:

```
âSᵛ□□ᵍ∞ᵸᵇ·ᵛᵒᵉf‡|ᵉ⁄ᵍ ⁚÷⋈ˢ〈 ㈳仔坟帙塹歐牋晾簿膠禋軌醾鮀乏佩勢呕噗塙婛犀幟愀扣摥晧桩橫汭湯
灱牳瑵卒硹穿籽繿庫芃蒫蚇袉誋貍躪邑銓钕隗顒骛耋麟ᵍᵒ ᵍ⁄ᵍ
□겭꺯낱늳뒵뚷뢹못벽뺿상싃쐹웆죉쭗첼컏탑퉇퓕훗\xD8\xD9□\xDE\xDF 〉 ⌉
ـشâSᵛ□□ᵍ∞ᵸᵇ·ᵛᵒᵉf‡|ᵉ⁄ᵍ ⁚÷⋈ˢ〈 ㈳仔坟帙塹歐牋晾簿膠禋軌醾鮀乏佩勢呕噗塙婛犀幟愀扣摥晧桩橫汭湯
```

灱牳瑵卒硹穿籽繿犀芃蒤蚇袉誋貍躪邑銓钕隗颙骛鍌麟ঙ ঁ ঁ
□뀭꺯낱늳뒵뚷뢹못벽뺿상싃쓅웆죉쯇쳍컏탑퉇풝훗\xD8\xD9□\xDE\xDF 〉 ⌉
(Additional content is omitted)

## 2.4.5OutputStreamWriter

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;

public class StreamWriter {

    public static void main(String[] args) {
//      try (OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream("output1_gb18030.txt"), "gb18030")) {
            try (OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream("output1_utf8.txt"), "utf8")) {

            String str = "你好！";
            osw.write(str);
            osw.flush();//osw.close();

        } catch (FileNotFoundException e) {
            System.out.println("The pathname does not exist.");
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            System.out.println("The Character Encoding is not supported.");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Failed or interrupted when doing the I/O operations");
            e.printStackTrace();
        }
    }
}
```

(1) Run above program, write “你好！” to output1_utf8.txt, charset is “utf8”;

(2) Modify the program, write “你好！” to output1_gb18030.txt, charset is “gb18030”;

(3) Open the output1_utf8.txt and output1_gb18030.txt in your notepad;

(4) Open the output1_utf8.txt and output1_gb18030.txt with a Hex Editor.

### 2.4.6 BufferWriter

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;

public class BufferWriter {

    public static void main(String[] args) {
        try (FileOutputStream fos = new FileOutputStream(new File("output2_gb18030.txt"));
                OutputStreamWriter osw = new OutputStreamWriter(fos, "gb18030");
                BufferedWriter bWriter = new BufferedWriter(osw);){
            bWriter.write("你好! \n");
//          bWriter.write(100);
            bWriter.write("100");
            bWriter.write(" 分 \n");
            bWriter.write("送给你! \n");
            bWriter.flush();//bWriter.close();

        } catch (FileNotFoundException e) {
            System.out.println("The pathname does not exist.");
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            System.out.println("The Character Encoding is not supported.");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Failed or interrupted when doing the I/O operations");
            e.printStackTrace();
        }
    }
}
```

(1) Run above program, open output1_gb18030.txt;

(2) Modify “100” to 100, open output1_gb18030.txt and see what happened;

(3) Modify above program, try to produce massive data and write to a file;

(4) Using OutputStreamWriter to write massive data to a file, compare the run time.

### 2.4.7 Scanning and Formatting

Programming I/O often involves translating to and from the neatly formatted data humans like to work with. To assist you with these chores, the Java platform provides two APIs. The scanner API breaks input into individual tokens associated with bits of data. The formatting API assembles data into nicely formatted, human-readable form.

Before, we usually use scanner to read data from console like this:

```
Scanner s = new Scanner( System.in );
s.nextDoulbe();
s.next();
```

传入的输入设备是 System.in

now we can also use it to read data from a file.

```
Scanner s = new Scanner(new BufferedReader(new FileReader("1.txt")));
```

Formatter example: 将内容按照某种格式写入文件中

```
Formatter formatter = new Formatter(new File("1.txt"));
formatter.format ("%s %f","Pi is", 3.0/7);
formatter.flush();
```

## 3 Charsets and Character Encoding

There are various ways for characters to be encoded as binary data. A particular encoding is known as a charset or character set . The encoding for charsets are specified by international standards organizations and have names such as “UTF-16”, “UTF-8,” and “ISO-8859-1”.

In UTF-16, characters are encoded as 16-bit UNICODE values; this is the character set that is **used internally by Java**. UTF-8 is another way of encoding UNICODE characters using 8 bits for common ASCII characters and longer codes for other characters. Both UTF-16 and UTF-8 use variable length encodings, UTF-16 uses either 2 or 4 bytes (instead of 1, 2, 3, or 4 bytes in UTF-8).

ISO-8859-1, is a widely used standard for Roman letters (ie English type letters and European variations), also known as “Latin-1,” is an 8-bit encoding that includes ASCII characters as well as certain accented characters that are used in several European languages.

### 3.1 Char vs binary value

Run the following code:

```
char c = '赵';
int value = c;
System.out.printf("%s\n",c);
System.out.printf("%X\n",value); 大写的十六进制
```

Observe the result.

### 3.2 Transform from different charset

Run the following code: Java的字符是基于UTF-16的标准编码的

```
    String str = "赵耀"; // UTF-16
    try
    {
        byte[] bytes1 = str.getBytes("GBK"); // or GBK
```

```
        for (byte b : bytes1) {
            System.out.printf("%2X ", b);
        }
        System.out.println();
        byte[] bytes2 = str.getBytes("UTF-16");
        for (byte b : bytes2) {
            System.out.printf("%02X ", b);
        }
        System.out.println();

        byte[] bytes3 = str.getBytes("UTF-16BE");
        for (byte b : bytes3) {
            System.out.printf("%02X ", b);
        }
        System.out.println();

        byte[] bytes4 = str.getBytes("UTF-16LE");
        for (byte b : bytes4) {
            System.out.printf("%02X ", b);
        }
        System.out.println();
    }catch(UnsupportedEncodingException e){

        e.printStackTrace();
    }
```

Observe the result.

PS: UTF-16:赵-8D75 耀-8000, GB:赵-D5D4 耀-D2AB

## 4 Some pitfalls

### 4.1 Sample 1 注意文件本身的编码格式

In `StreamReader`(2.4.2 InputStreamReader), try to change the following line:

```
InputStreamReader isr = new InputStreamReader(fis, "gb18030");
```

To

```
InputStreamReader isr = new InputStreamReader(fis, "utf8");
```

Observe the result.

### 4.2 Sample 2

Try to run the following code:

```
public class SurrogatePairsTest {

    public static void main(String[] args) {
```

超过16bit会使用两个char进行存储

```
        String s=String.valueOf(Character.toChars(0x10437));
        System.out.print(s);
        System.out.println(s.charAt(0));

        char[]chars=s.toCharArray();
        for(char c:chars){
            System.out.format("%x",(short)c);
        }
    }
}
```

这时我们可以使用String去打印这个超过16bit的字符

Observe the result and explain why the output of `s` is not the same as `s.charAt(0)` ? Why `0x10437` could be converted to `0xd801dc37` ?

**Answer:**

UTF-16 is used internally by Java, and Java primitive type `char` is 16 bits wide. When a Unicode character is with code above 0xFFFF, is encoded in UTF-16 by pairs of 16-bit code units called **surrogate pairs**.

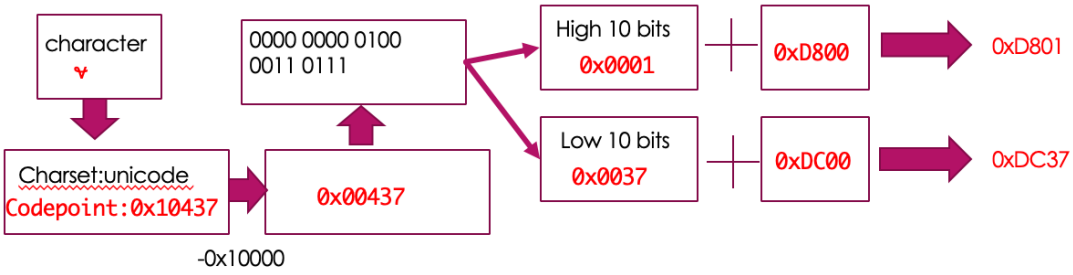
`0x10437 to 0xd801dc37`

Step1: 0x10437 minus 0x10000 gives 0x00437, binary 0000 0000 0100 0011 0111(0x00437)

Step2: Partition its upper and lower 10 bit values (binary) :0000000001 and 0000110111

Step3: Add 0xD800 to the upper value to form the higher part: 0xD800 + 0x0001 = 0xD801

Step4: Add 0xDC00 to the lower value to form a lower part: 0xDC00 + 0x0037 = 0xDC37.




Hint: Don't use notepad please use Notepad++ / VS Code/Sublime Text and other software that can handle multiple encodings easily!

# Reference

https://zh.wikipedia.org/wiki/Unicode%E5%AD%97%E7%AC%A6%E5%B9%B3%E9%9D%A2%E6%98%A0%E5%B0%84

https://unicode-table.com/cn/blocks/cjk-unified-ideographs/

https://www.qqxiuzi.cn/bianma/zifuji.php

https://www.jianshu.com/p/ad4bff4d9fa3

https://docs.oracle.com/javase/8/docs/technotes/guides/intl/overview.html

http://blog.51cto.com/cnn237111/1080628

https://docs.oracle.com/javase/tutorial/essential/io/index.html

https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html

https://docs.oracle.com/javase/7/docs/api/index.html?java/nio/package-summary.html