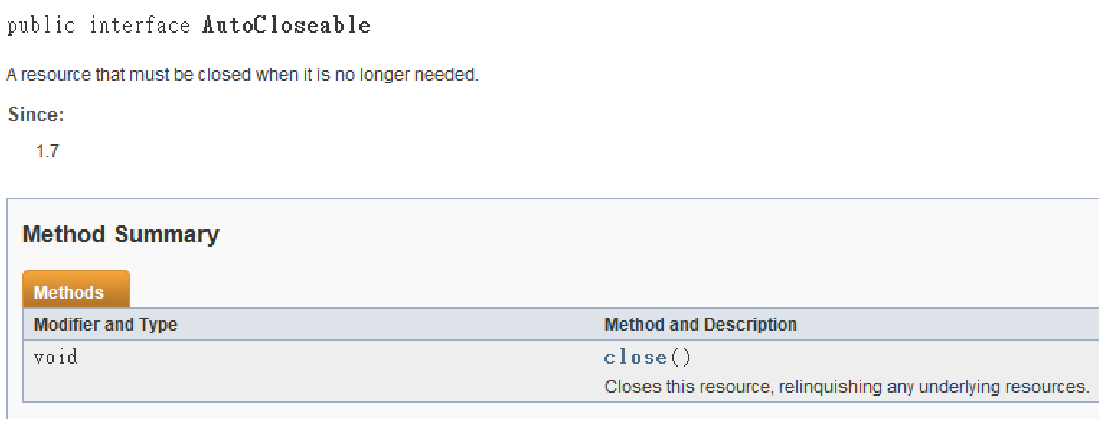
自動嘗試關閉資源 JDK7 新功能

try-with-resources 述句

1. try-with-resources 可以讓我們在 try 敘述句中宣告 一 到 數項資源，當我們說某個物件是一項資源時，意味著該物件實作了 java.lang.AutoCloseable 或是它的子介面 java.lang.Closeable，並且當程式不再使用它們時，我們需要將其關閉。
   try 新增的 語法 可確保每項資源在區段結束時都會被關閉。
2. AutoCloseable 介面 ➔AutoCloseable 是 JDK7 新增的介面，只有定義 close()方法
3. Closeable 介面繼承 AutoCloseable 介面➔java.io.Closeable 繼承 java.lang.AutoCloseable 的介面
4. 就兩個介面的宣告上來說，幾乎沒什麼差別，唯一的差別就是 java.io.Closeable 的 close 方法宣告會丟出 IOException 例外，而 java.lang.AutoCloseable 的 close 方法宣告會丟出 Exception 例外
5. JDK7 的嘗試關閉資源語法可套用的物件，都必須實作 AutoCloseable 介面 或 Closeable 介面
   (1). 位元串流➔InputStream，OutputStream 實作 java.io.Closeable
   (2). 字元串流➔FileReader，FileWriter， BufferReader，BufferWriter，PrintWriter 實作 AutoCloseable
   (3). JDBC➔Connection，Statement，ResultSet 實作 AutoCloseable
6. 嘗試關閉資源語法也可以同時關閉兩個以上的物件資源，只要中間以分號區隔，在 try 的括號中，越後面撰寫的物件資源會越早被關閉，每個 AutoCloseable 物件，都獨立使用一個 try，catch，filally，括號中 越後面撰寫的物件，會是在越內層的 try，catch，finally 中

http://docs.oracle.com/javase/7/docs/api/java/lang/AutoCloseable.html

http://docs.oracle.com/javase/7/docs/api/java/io/Closeable.html



```
public interface AutoCloseable
```

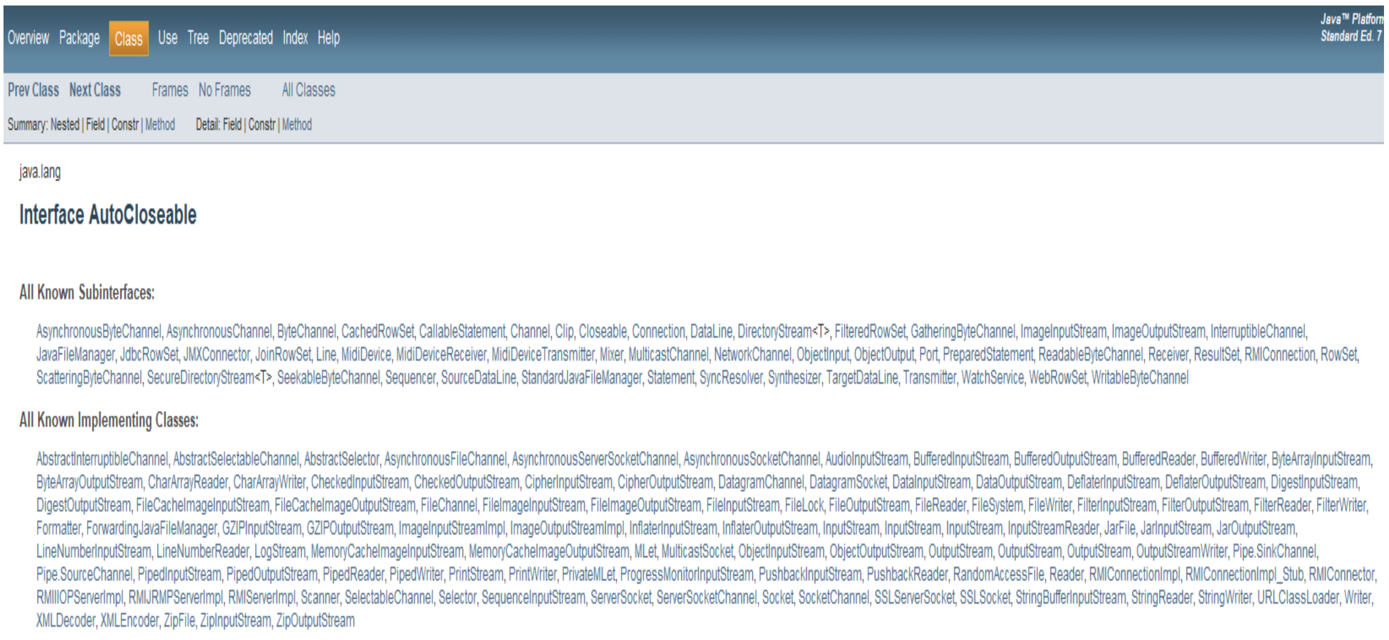
A resource that must be closed when it is no longer needed.

Since:
1.7

Method Summary

Methods

Modifier and Type	Method and Description
void	close() Closes this resource, relinquishing any underlying resources.



Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames All Classes
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

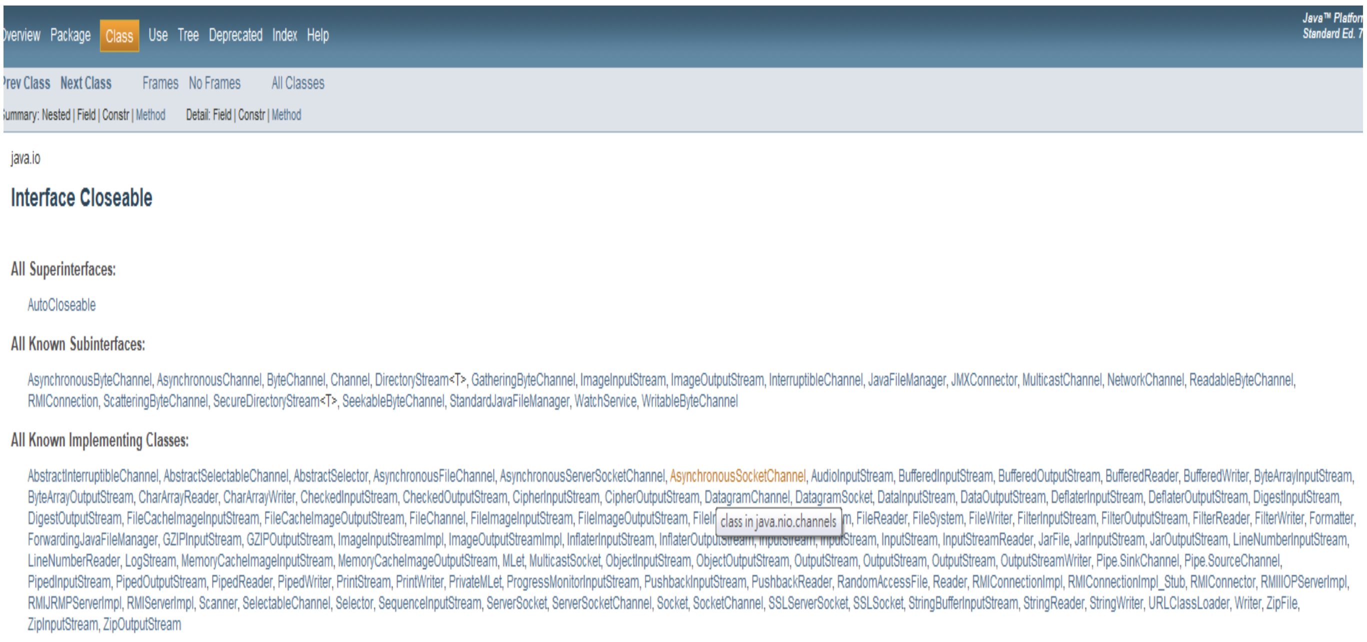
Interface AutoCloseable

All Known Subinterfaces:

AsynchronousByteChannel, AsynchronousChannel, ByteChannel, CachedRowSet, CallableStatement, Channel, Clip, Closeable, Connection, DataLine, DirectoryStream<T>, FilteredRowSet, GatheringByteChannel, ImageInputStream, ImageOutputStream, InterruptibleChannel, JavaFileManager, JdbcRowSet, JMXConnector, JoinRowSet, Line, MidiDevice, MidiDeviceReceiver, MidiDeviceTransmitter, Mixer, MulticastChannel, NetworkChannel, ObjectInput, ObjectOutput, Port, PreparedStatement, ReadableByteChannel, Receiver, ResultSet, RMIConnection, RowSet, ScatteringByteChannel, SecureDirectoryStream<T>, SeekableByteChannel, Sequencer, SourceDataLine, StandardJavaFileManager, Statement, SyncResolver, Synthesizer, TargetDataLine, Transmitter, WatchService, WebRowSet, WritableByteChannel

All Known Implementing Classes:

AbstractInterruptibleChannel, AbstractSelectableChannel, AbstractSelector, AsynchronousFileChannel, AsynchronousServerSocketChannel, AsynchronousSocketChannel, AudioInputStream, BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter, ByteArrayInputStream, ByteArrayOutputStream, CharArrayReader, CharArrayWriter, CheckedInputStream, CheckedOutputStream, CipherInputStream, CipherOutputStream, DatagramChannel, DatagramSocket, DataInputStream, DataOutputStream, DeflaterInputStream, DeflaterOutputStream, DigestInputStream, DigestOutputStream, FileCacheImageInputStream, FileCacheImageOutputStream, FileChannel, FileImageInputStream, FileImageOutputStream, FileInputStream, FileLock, FileOutputStream, FileReader, FileSystem, FileWriter, FilterInputStream, FilterOutputStream, FilterReader, FilterWriter, Formatter, ForwardingJavaFileManager, GZIPInputStream, GZIPOutputStream, ImageInputStreamImpl, ImageOutputStreamImpl, InflaterInputStream, InflaterOutputStream, InputStream, InputStream, InputStream, InputStreamReader, JarFile, JarInputStream, JarOutputStream, LineNumberInputStream, LineNumberReader, LogStream, MemoryCacheImageInputStream, MemoryCacheImageOutputStream, MLet, MulticastSocket, ObjectInputStream, ObjectOutputStream, OutputStream, OutputStream, OutputStream, OutputStreamWriter, Pipe.SinkChannel, Pipe.SourceChannel, PipedInputStream, PipedOutputStream, PipedReader, PipedWriter, PrintStream, PrintWriter, PrivateMLet, ProgressMonitorInputStream, PushbackInputStream, PushbackReader, RandomAccessFile, Reader, RMIConnectionImpl, RMIConnectionImpl_Stub, RMIConnector, RMIIIOPServerImpl, RMIJRMPServerImpl, RMIServerImpl, Scanner, SelectableChannel, Selector, SequenceInputStream, ServerSocket, ServerSocketChannel, Socket, SocketChannel, SSLServerSocket, SSLSocket, StringBufferInputStream, StringReader, StringWriter, URLClassLoader, Writer, XMLDecoder, XMLEncoder, ZipFile, ZipInputStream, ZipOutputStream


Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames All Classes
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.io

Interface Closeable

All Superinterfaces:

AutoCloseable

All Known Subinterfaces:

AsynchronousByteChannel, AsynchronousChannel, ByteChannel, Channel, DirectoryStream<T>, GatheringByteChannel, ImageInputStream, ImageOutputStream, InterruptibleChannel, JavaFileManager, JMXConnector, MulticastChannel, NetworkChannel, ReadableByteChannel, RMIConnection, ScatteringByteChannel, SecureDirectoryStream<T>, SeekableByteChannel, StandardJavaFileManager, WatchService, WritableByteChannel

All Known Implementing Classes:

AbstractInterruptibleChannel, AbstractSelectableChannel, AbstractSelector, AsynchronousFileChannel, AsynchronousServerSocketChannel, AsynchronousSocketChannel, AudioInputStream, BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter, ByteArrayInputStream, ByteArrayOutputStream, CharArrayReader, CharArrayWriter, CheckedInputStream, CheckedOutputStream, CipherInputStream, CipherOutputStream, DatagramChannel, DatagramSocket, DataInputStream, DataOutputStream, DeflaterInputStream, DeflaterOutputStream, DigestInputStream, DigestOutputStream, FileCacheImageInputStream, FileCacheImageOutputStream, FileChannel, FileImageInputStream, FileImageOutputStream, FileI class in java.nio.channels m, FileReader, FileSystem, FileWriter, FilterInputStream, FilterOutputStream, FilterReader, FilterWriter, Formatter, ForwardingJavaFileManager, GZIPInputStream, GZIPOutputStream, ImageInputStreamImpl, ImageOutputStreamImpl, InflaterInputStream, InflaterOutputStream, InputStream, InputStream, InputStream, InputStreamReader, JarFile, JarInputStream, JarOutputStream, LineNumberInputStream, LineNumberReader, LogStream, MemoryCacheImageInputStream, MemoryCacheImageOutputStream, MLet, MulticastSocket, ObjectInputStream, ObjectOutputStream, OutputStream, OutputStream, OutputStream, OutputStreamWriter, Pipe.SinkChannel, Pipe.SourceChannel, PipedInputStream, PipedOutputStream, PipedReader, PipedWriter, PrintStream, PrintWriter, PrivateMLet, ProgressMonitorInputStream, PushbackInputStream, PushbackReader, RandomAccessFile, Reader, RMIConnectionImpl, RMIConnectionImpl_Stub, RMIConnector, RMIIIOPServerImpl, RMIJRMPServerImpl, RMIServerImpl, Scanner, SelectableChannel, Selector, SequenceInputStream, ServerSocket, ServerSocketChannel, Socket, SocketChannel, SSLServerSocket, SSLSocket, StringBufferInputStream, StringReader, StringWriter, URLClassLoader, Writer, ZipFile, ZipInputStream, ZipOutputStream

```java
//自定類別要具有自動關閉資源的能力 ，必須實作 AutoCloseable介面
class Resource implements AutoCloseable {

    public void doSome() {
        System.out.println("作一些事");
    }

    @Override
    public void close() throws Exception { //實作close()方法
        System.out.println("資源被關閉");
    }
}

public class Auto11 {

    public static void main(String[] args) {

        try (Resource res = new Resource()) {
            res.doSome();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
class ResourceSome implements AutoCloseable {

    public void doSome() {
        System.out.println("作一些事_1");
    }

    @Override
    public void close() throws Exception {
        System.out.println("資源Some被關閉_1");
    }
}

class ResourceOther implements AutoCloseable {

    public void doOther() {
        System.out.println("作其它事_2");
    }

    @Override
    public void close() throws Exception {
        System.out.println("資源Other被關閉_2");
    }
}

public class Auto12 {

    public static void main(String[] args) {

        try (ResourceSome some = new ResourceSome();
                ResourceOther other = new ResourceOther()) {
            some.doSome();
            other.doOther();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
作一些事
資源被關閉
```

```
作一些事_1
作其它事_2
資源Other被關閉_2
資源Some被關閉_1
```

檔案操作 I/O

1. 字元串流➔FileReader，FileWriter，BufferedReader，BufferedWriter，PrintWriter➔實作 AutoCloseable 介面
2. 位元串流➔InputStream 或 OutputStream➔實作 java.io.Closeable 介面 是 AutoCloseable 的子介面

字元串流➔相關類別

1. File➔API 說 File 類別是 “關於檔案和目錄名稱的抽象表示” ，File 類別實際上不是用來讀取或寫入資料，它是在更高層次的地方運作，包含 : 建立空檔案，尋找檔案，建立目錄和操作路徑
2. FileReader➔這個類別是用來讀取文字檔案，它的 read()函式是非常低階的，允許你讀取單一字元，整個字串流或固定數量的字元，FileReader 通常被更高階的 Reader 物件像 BufferedReader 所包覆，這樣做可以增加效率，或提供更方便的方式來操作資料
3. BufferedReader➔這個類別讓低階的 Reader 類別，例如 FileReader，可以更方便和更容易地被使用，和 FileReader 比較一下, BufferedReader 可以從檔案一次讀取比較大區塊的資料，並且保留在暫存區內，當你要求下一個字元或下一行的資料，就會從暫存區內讀取，這可以減少讀取檔案的次數，另外 BufferedReader 也提供更方便的函式，像 readLine()可以取得檔案的下一行
4. FileWriter➔這個類別被用來將資料寫入文字檔案，它的 write()函式允許你將字元或 String 寫進檔案，FileWriter 通常被更高階的 Writer 物件，像 BufferedWriter 或 PrintWriter 所包覆，這樣就可以提供更好的效率，和更高階且彈性的函式來寫入資料
5. BufferedWriter➔這個類別被用來讓低階類別，像 FileWriter,可以更有效率且更容易地被使用，和 FileWriter 相比較，BufferedWriter 可以一次寫入較大區塊的資料進入檔案，進而減少，速度較慢的檔案寫入的動作的次數，另外，BufferedWriter 類別提供 newLine()函式，供你更容易建立平台時特有的換行符號
6. PrintWriter➔這個類別已經在 Java5 被顯著地加強，因為新的函式和建構子讓你可以直接使用 PrintWriter 來寫出資料，而不需要再事先建立 FileWriter 或 BufferedWriter，像 format()，print() 和 append()這些新函式也加強了 PrintWriter 的彈性和功能

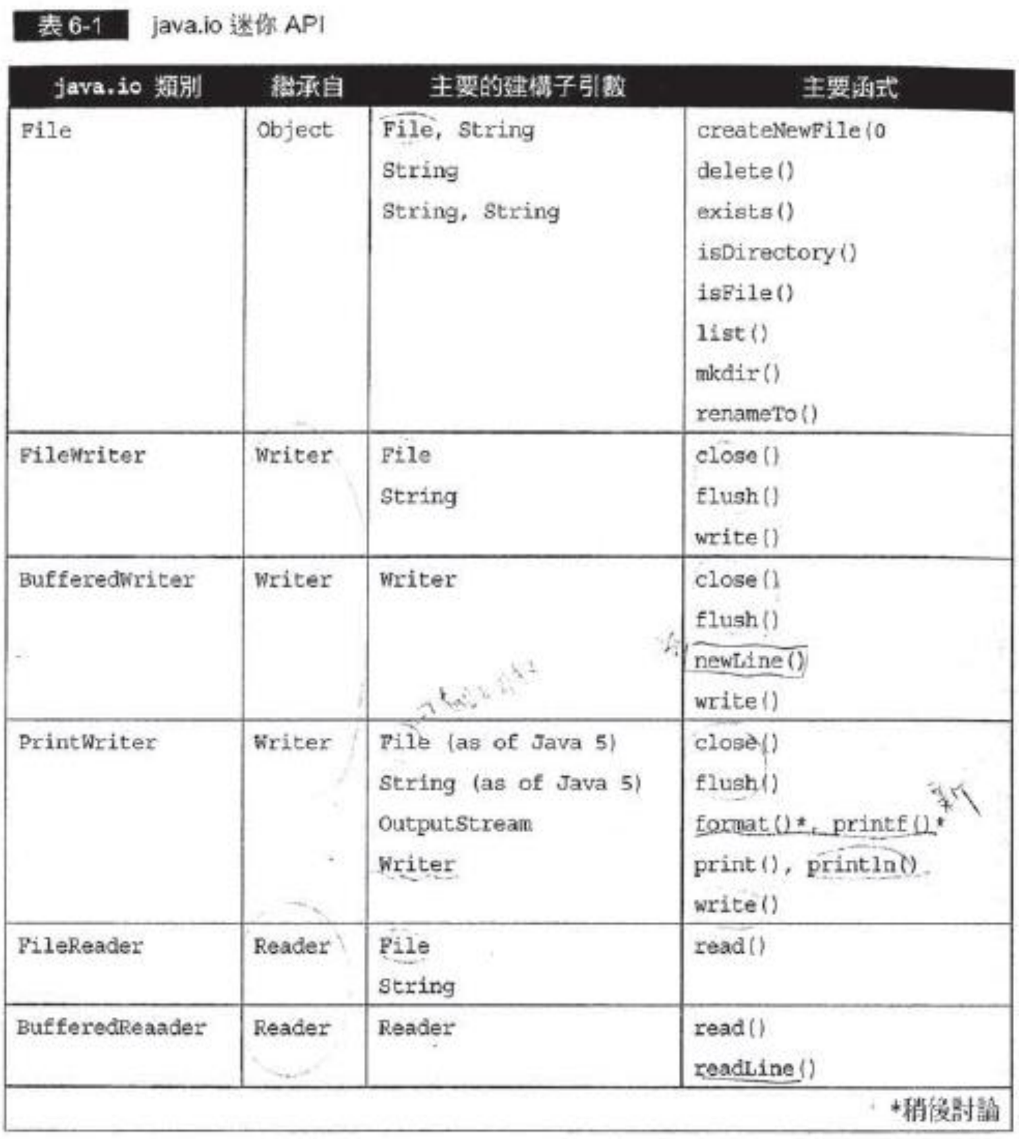


表 6-1 java.io 迷你 API

java.io 類別	繼承自	主要的建構子引數	主要函式

File	Object	File, String
		String
		String, String
			createNewFile(0
			delete()
			exists()
			isDirectory()
			isFile()
			list()
			mkdir()
			renameTo()

FileWriter	Writer	File
		String
			close()
			flush()
			write()

BufferedWriter	Writer	Writer	close()
			flush()
			newLine()
			write()

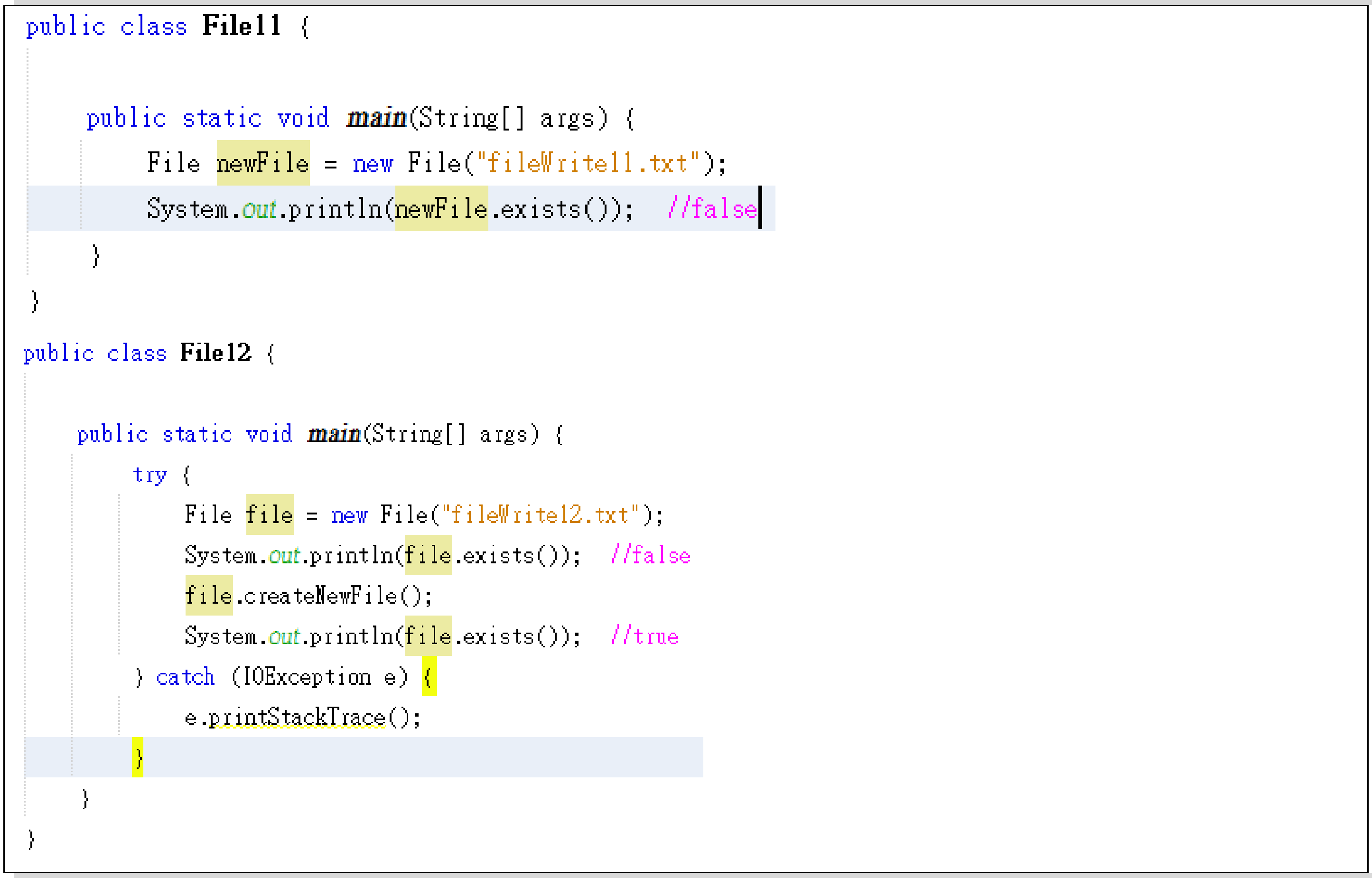
PrintWriter	Writer	File (as of Java 5)
		String (as of Java 5)
		OutputStream
		Writer
			close()
			flush()
			format()*, printf()*
			print(), println()
			write()

FileReader	Reader	File String	read()

BufferedReaader	Reader	Reader	read()
			readLine()

			*稍後討論

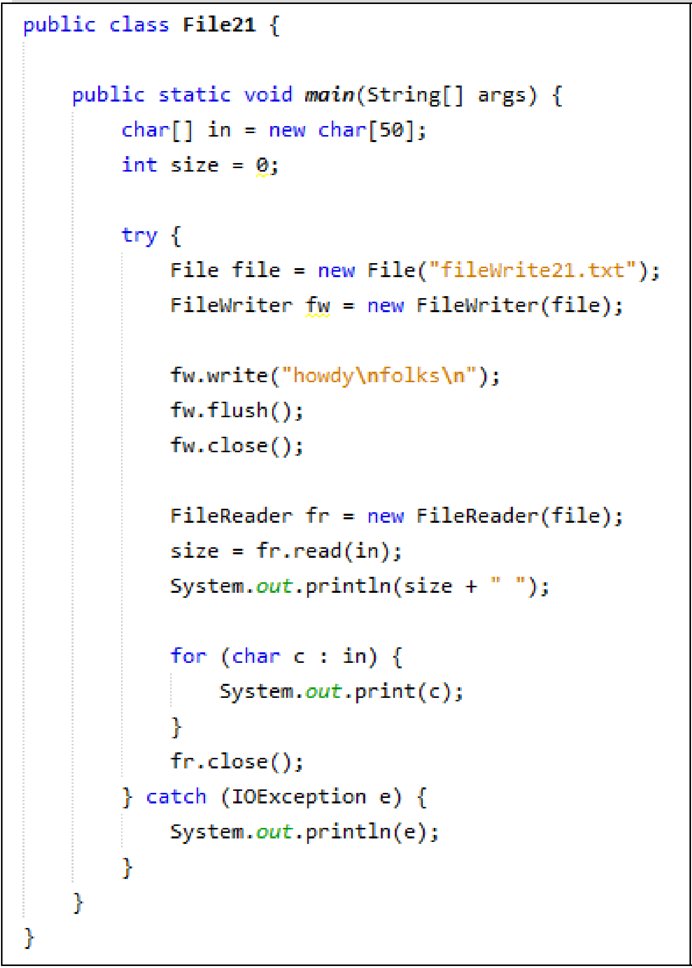
使用 File 建立檔案



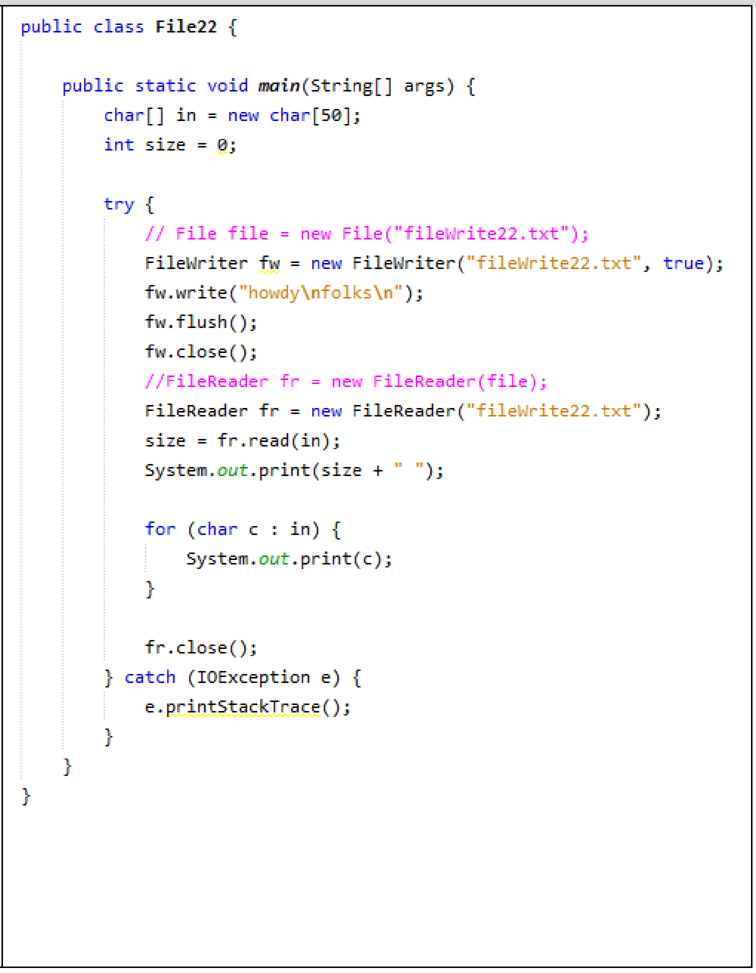
```
public class File11 {

    public static void main(String[] args) {
        File newFile = new File("fileWrite11.txt");
        System.out.println(newFile.exists());  //false
    }
}
```

```
public class File12 {

    public static void main(String[] args) {
        try {
            File file = new File("fileWrite12.txt");
            System.out.println(file.exists());  //false
            file.createNewFile();
            System.out.println(file.exists());  //true
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

使用 FileReader 和 FileWriter



```
public class File21 {

    public static void main(String[] args) {
        char[] in = new char[50];
        int size = 0;

        try {
            File file = new File("fileWrite21.txt");
            FileWriter fw = new FileWriter(file);

            fw.write("howdy\nfolks\n");
            fw.flush();
            fw.close();

            FileReader fr = new FileReader(file);
            size = fr.read(in);
            System.out.println(size + " ");

            for (char c : in) {
                System.out.print(c);
            }
            fr.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```



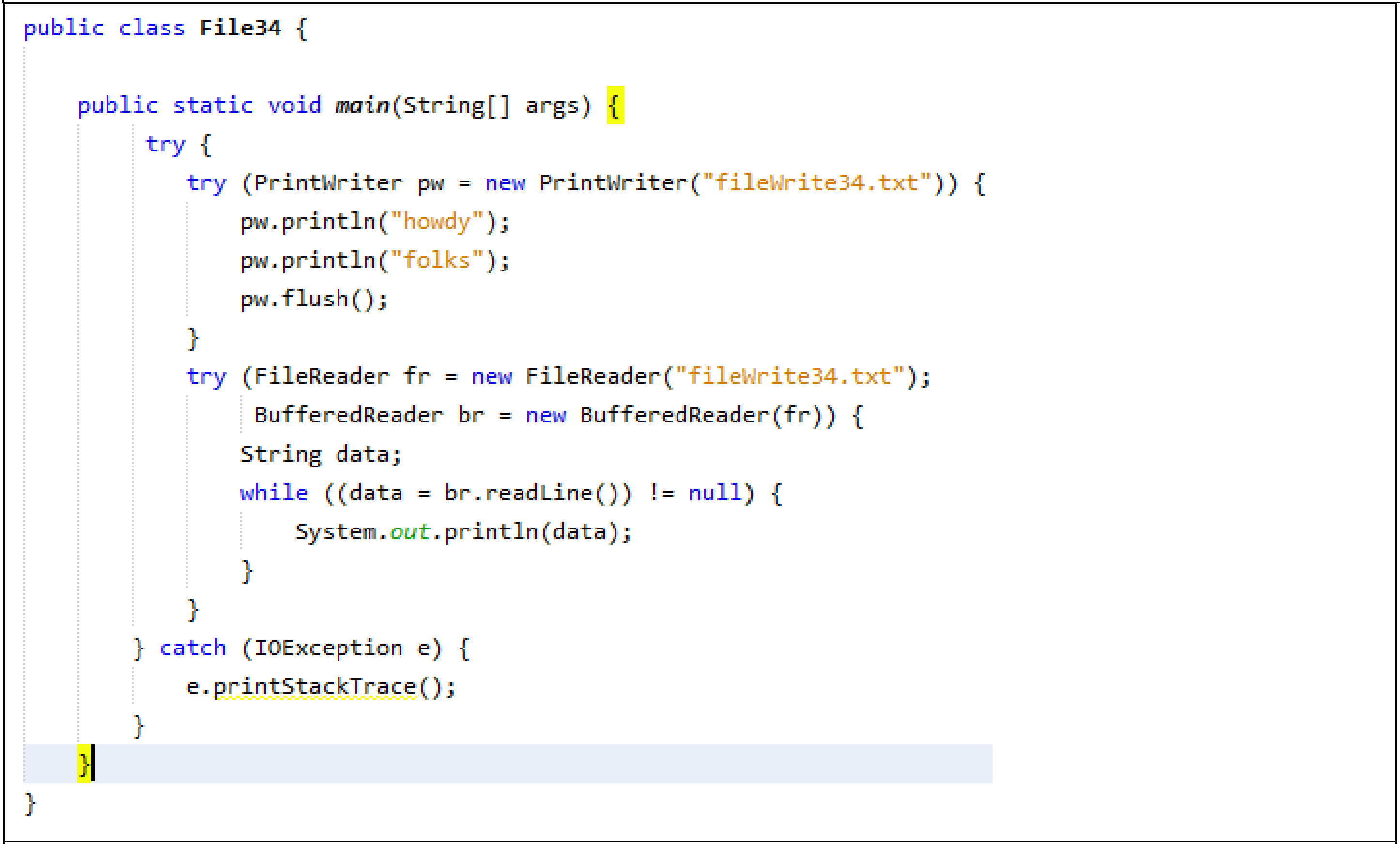
```
public class File22 {

    public static void main(String[] args) {
        char[] in = new char[50];
        int size = 0;

        try {
            // File file = new File("fileWrite22.txt");
            FileWriter fw = new FileWriter("fileWrite22.txt", true);
            fw.write("howdy\nfolks\n");
            fw.flush();
            fw.close();
            //FileReader fr = new FileReader(file);
            FileReader fr = new FileReader("fileWrite22.txt");
            size = fr.read(in);
            System.out.print(size + " ");

            for (char c : in) {
                System.out.print(c);
            }

            fr.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

組合使用 I/O 類別

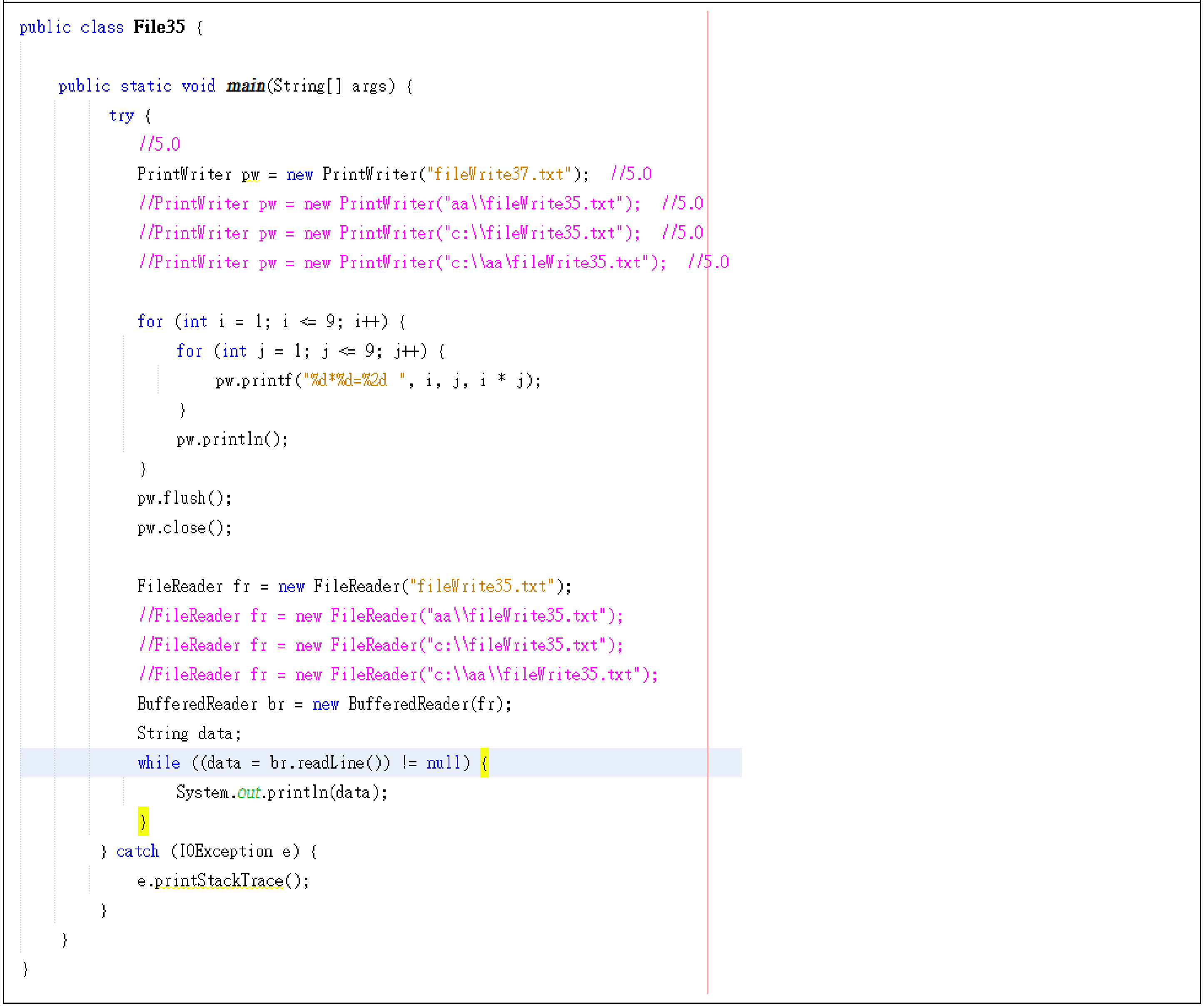
```java
public class File31 {

    public static void main(String[] args) {
        try {
          File file = new File("fileWrite31.txt");
          FileWriter fw = new FileWriter(file);
          BufferedWriter bw = new BufferedWriter(fw);

          bw.write("howdy");
          bw.newLine();
          bw.write("folks");
          bw.newLine();

          bw.flush();
          bw.close();

          FileReader fr = new FileReader(file);
          BufferedReader br = new BufferedReader(fr);

          String data;
          while ((data = br.readLine()) != null) {
              System.out.println(data);
          }
        } catch (IOException e) {
          e.printStackTrace();
        }
    }
}
```

```java
public class File32 {

    public static void main(String[] args) {
        try {
          File file = new File("fileWrite32.txt");
          PrintWriter pw = new PrintWriter(file);  //5.0

          pw.println("howdy");
          pw.println("folks");
          pw.flush();
          pw.close();

          FileReader fr = new FileReader(file);
          BufferedReader br = new BufferedReader(fr);

          String data;
          while ((data = br.readLine()) != null) {
              System.out.println(data);
          }
        } catch (IOException e) {
          e.printStackTrace();
        }
    }
}
```

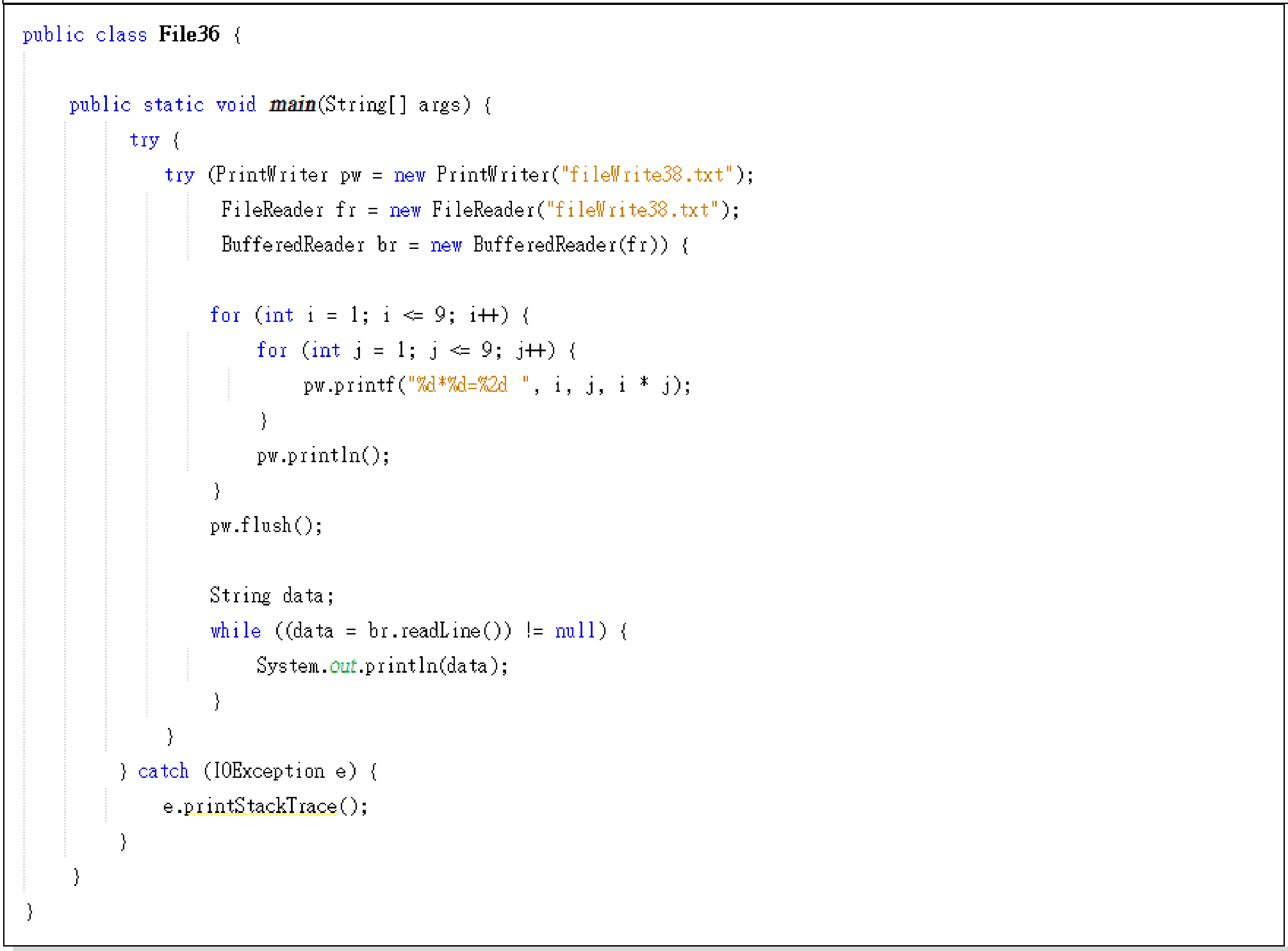
```java
public class File33 {

    public static void main(String[] args) {
        try {
          //5.0
          PrintWriter pw = new PrintWriter("fileWrite33.txt");

          pw.println("howdy");
          pw.println("folks");
          pw.flush();
          pw.close();

          FileReader fr = new FileReader("fileWrite33.txt");
          BufferedReader br = new BufferedReader(fr);

          String data;
          while ((data = br.readLine()) != null) {
              System.out.println(data);
          }
        } catch (IOException e) {
          e.printStackTrace();
        }
    }
}
```



```
public class File34 {

    public static void main(String[] args) {
        try {
            try (PrintWriter pw = new PrintWriter("fileWrite34.txt")) {
                pw.println("howdy");
                pw.println("folks");
                pw.flush();
            }
            try (FileReader fr = new FileReader("fileWrite34.txt");
                 BufferedReader br = new BufferedReader(fr)) {
                String data;
                while ((data = br.readLine()) != null) {
                    System.out.println(data);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



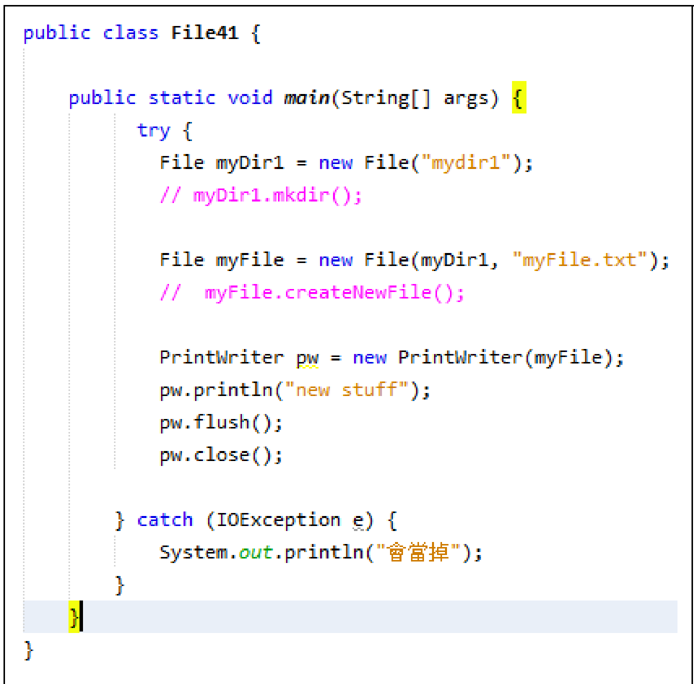
```
public class File35 {

    public static void main(String[] args) {
        try {
            //5.0
            PrintWriter pw = new PrintWriter("fileWrite37.txt");  //5.0
            //PrintWriter pw = new PrintWriter("aa\\fileWrite35.txt");  //5.0
            //PrintWriter pw = new PrintWriter("c:\\fileWrite35.txt");  //5.0
            //PrintWriter pw = new PrintWriter("c:\\aa\fileWrite35.txt");  //5.0

            for (int i = 1; i <= 9; i++) {
                for (int j = 1; j <= 9; j++) {
                    pw.printf("%d*%d=%2d  ", i, j, i * j);
                }
                pw.println();
            }
            pw.flush();
            pw.close();

            FileReader fr = new FileReader("fileWrite35.txt");
            //FileReader fr = new FileReader("aa\\fileWrite35.txt");
            //FileReader fr = new FileReader("c:\\fileWrite35.txt");
            //FileReader fr = new FileReader("c:\\aa\\fileWrite35.txt");
            BufferedReader br = new BufferedReader(fr);
            String data;
            while ((data = br.readLine()) != null) {
                System.out.println(data);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



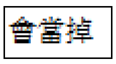
```java
public class File36 {

    public static void main(String[] args) {
        try {
            try (PrintWriter pw = new PrintWriter("fileWrite38.txt");
                 FileReader fr = new FileReader("fileWrite38.txt");
                 BufferedReader br = new BufferedReader(fr)) {

                for (int i = 1; i <= 9; i++) {
                    for (int j = 1; j <= 9; j++) {
                        pw.printf("%d*%d=%2d ", i, j, i * j);
                    }
                    pw.println();
                }
                pw.flush();

                String data;
                while ((data = br.readLine()) != null) {
                    System.out.println(data);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

操作檔案和目錄

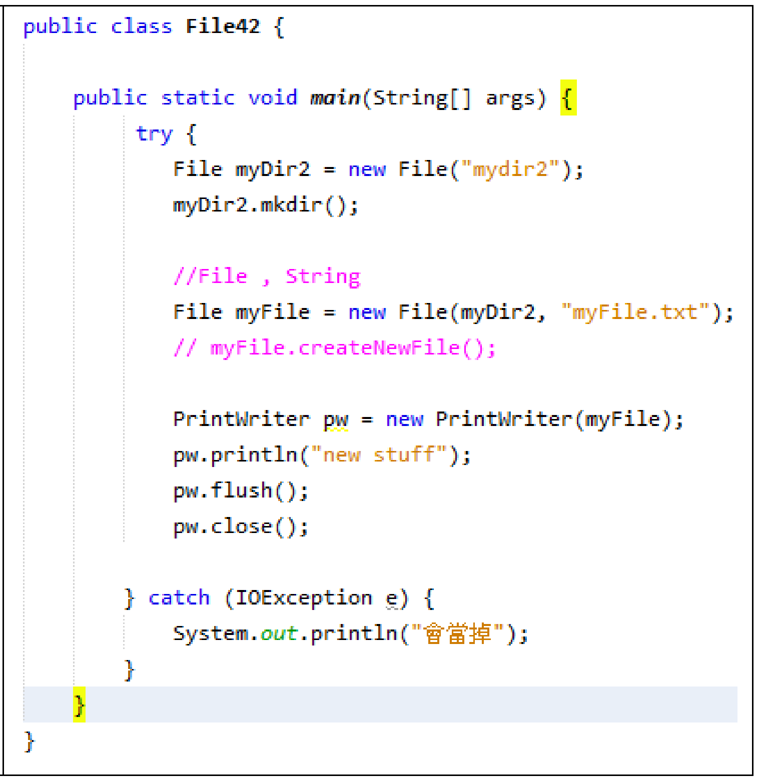
1. File 類別可以用來建立檔案和目錄，另外，File 的函式可以用來刪除檔案，修改檔案名稱，判斷檔案是否存在，建立暫存檔，修改檔案屬性，和區分它到底是檔案還是目錄
2. 建立目錄的時候要小心 ！ 建構一個 Reader 或 Writer 時，如果檔案不存在，JVM 就會自動幫你建立一個檔案，但這樣的事情並不會發生在目錄身上



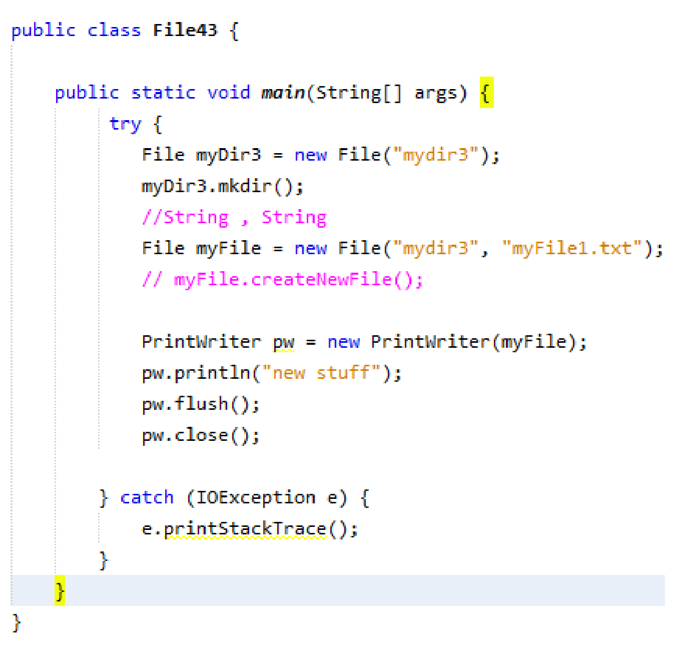
```java
public class File41 {

    public static void main(String[] args) {
        try {
            File myDir1 = new File("mydir1");
            // myDir1.mkdir();

            File myFile = new File(myDir1, "myFile.txt");
            //  myFile.createNewFile();

            PrintWriter pw = new PrintWriter(myFile);
            pw.println("new stuff");
            pw.flush();
            pw.close();

        } catch (IOException e) {
            System.out.println("會當掉");
        }
    }
}
```



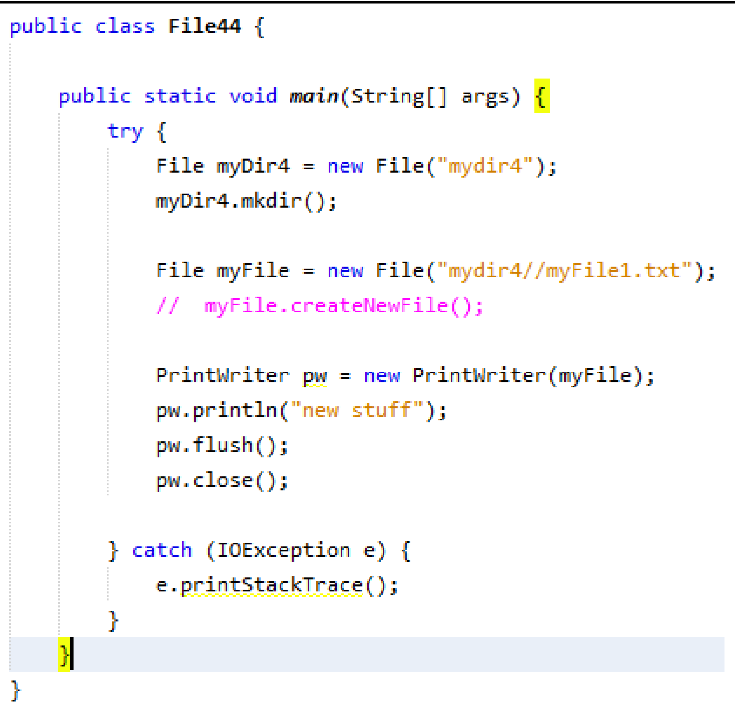
```
會當掉
```



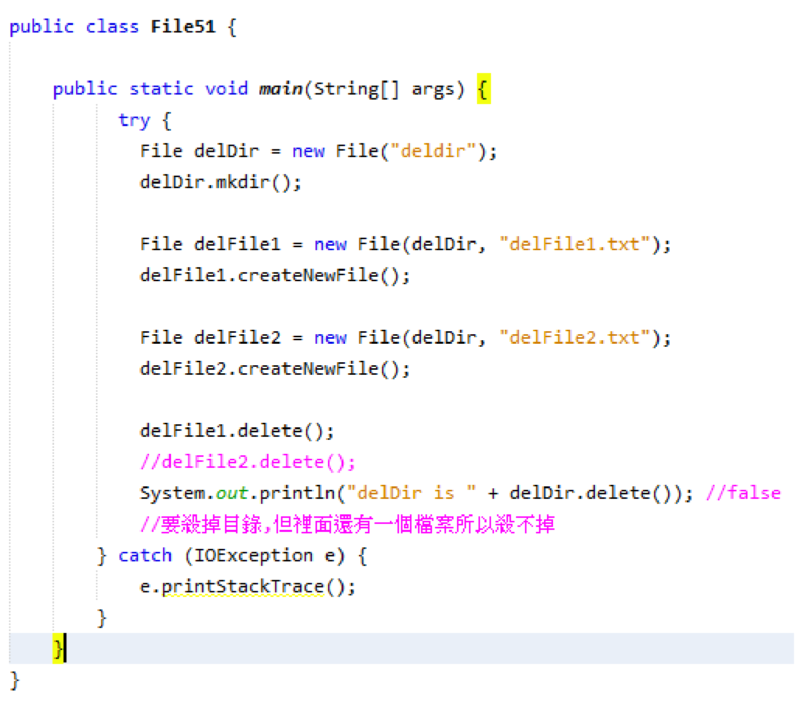
```java
public class File42 {

    public static void main(String[] args) {
        try {
            File myDir2 = new File("mydir2");
            myDir2.mkdir();

            //File , String
            File myFile = new File(myDir2, "myFile.txt");
            // myFile.createNewFile();

            PrintWriter pw = new PrintWriter(myFile);
            pw.println("new stuff");
            pw.flush();
            pw.close();

        } catch (IOException e) {
            System.out.println("會當掉");
        }
    }
}
```



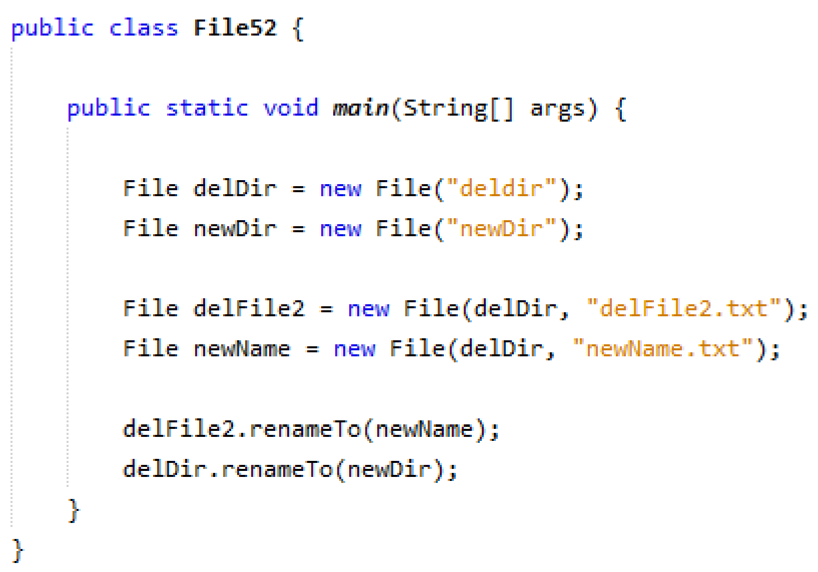
```java
public class File43 {

    public static void main(String[] args) {
        try {
            File myDir3 = new File("mydir3");
            myDir3.mkdir();
            //String , String
            File myFile = new File("mydir3", "myFile1.txt");
            // myFile.createNewFile();

            PrintWriter pw = new PrintWriter(myFile);
            pw.println("new stuff");
            pw.flush();
            pw.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



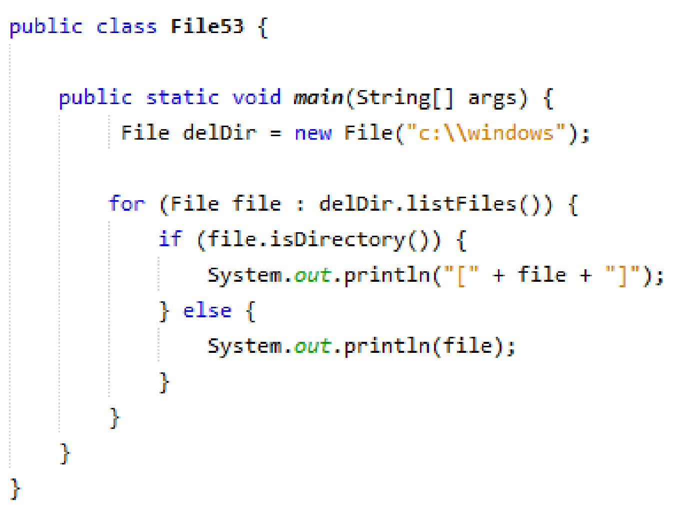
```java
public class File44 {

    public static void main(String[] args) {
        try {
            File myDir4 = new File("mydir4");
            myDir4.mkdir();

            File myFile = new File("mydir4//myFile1.txt");
            //  myFile.createNewFile();

            PrintWriter pw = new PrintWriter(myFile);
            pw.println("new stuff");
            pw.flush();
            pw.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



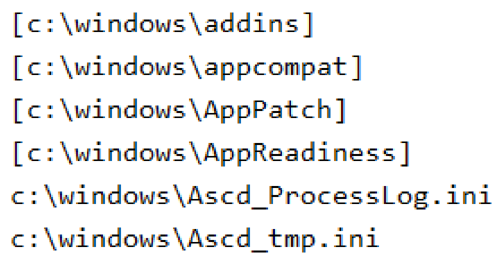
```java
public class File51 {

    public static void main(String[] args) {
        try {
            File delDir = new File("deldir");
            delDir.mkdir();

            File delFile1 = new File(delDir, "delFile1.txt");
            delFile1.createNewFile();

            File delFile2 = new File(delDir, "delFile2.txt");
            delFile2.createNewFile();

            delFile1.delete();
            //delFile2.delete();
            System.out.println("delDir is " + delDir.delete()); //false
            //要殺掉目錄,但裡面還有一個檔案所以殺不掉
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



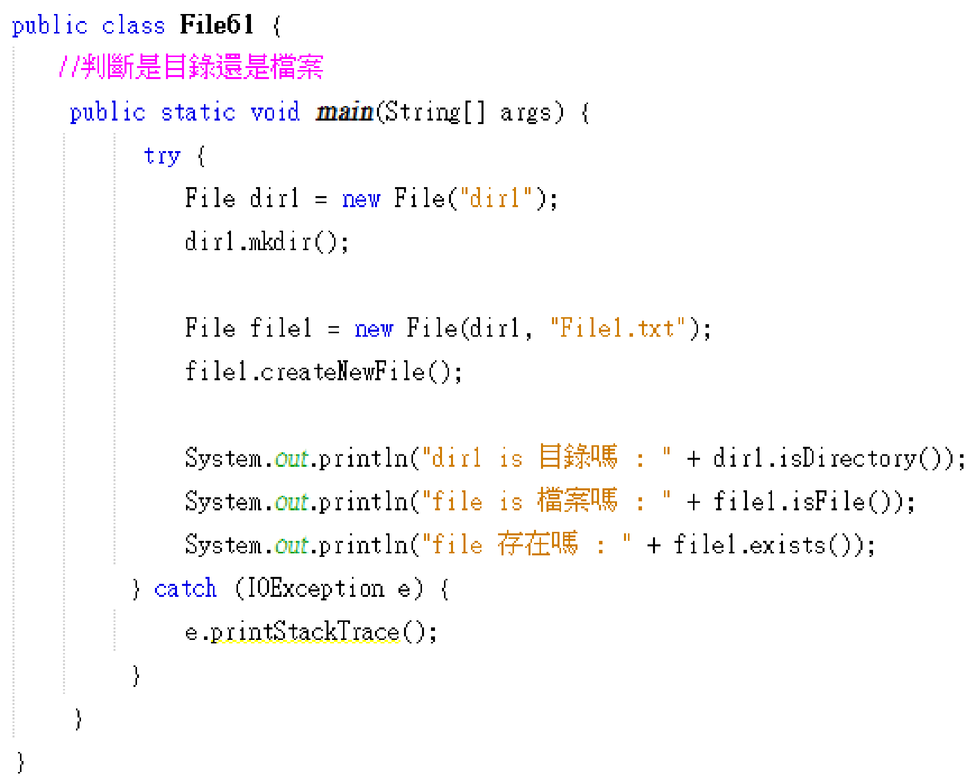
```java
public class File52 {

    public static void main(String[] args) {

        File delDir = new File("deldir");
        File newDir = new File("newDir");

        File delFile2 = new File(delDir, "delFile2.txt");
        File newName = new File(delDir, "newName.txt");

        delFile2.renameTo(newName);
        delDir.renameTo(newDir);
    }
}
```



```java
public class File53 {

    public static void main(String[] args) {
        File delDir = new File("c:\\windows");

        for (File file : delDir.listFiles()) {
            if (file.isDirectory()) {
                System.out.println("[" + file + "]");
            } else {
                System.out.println(file);
            }
        }
    }
}
```



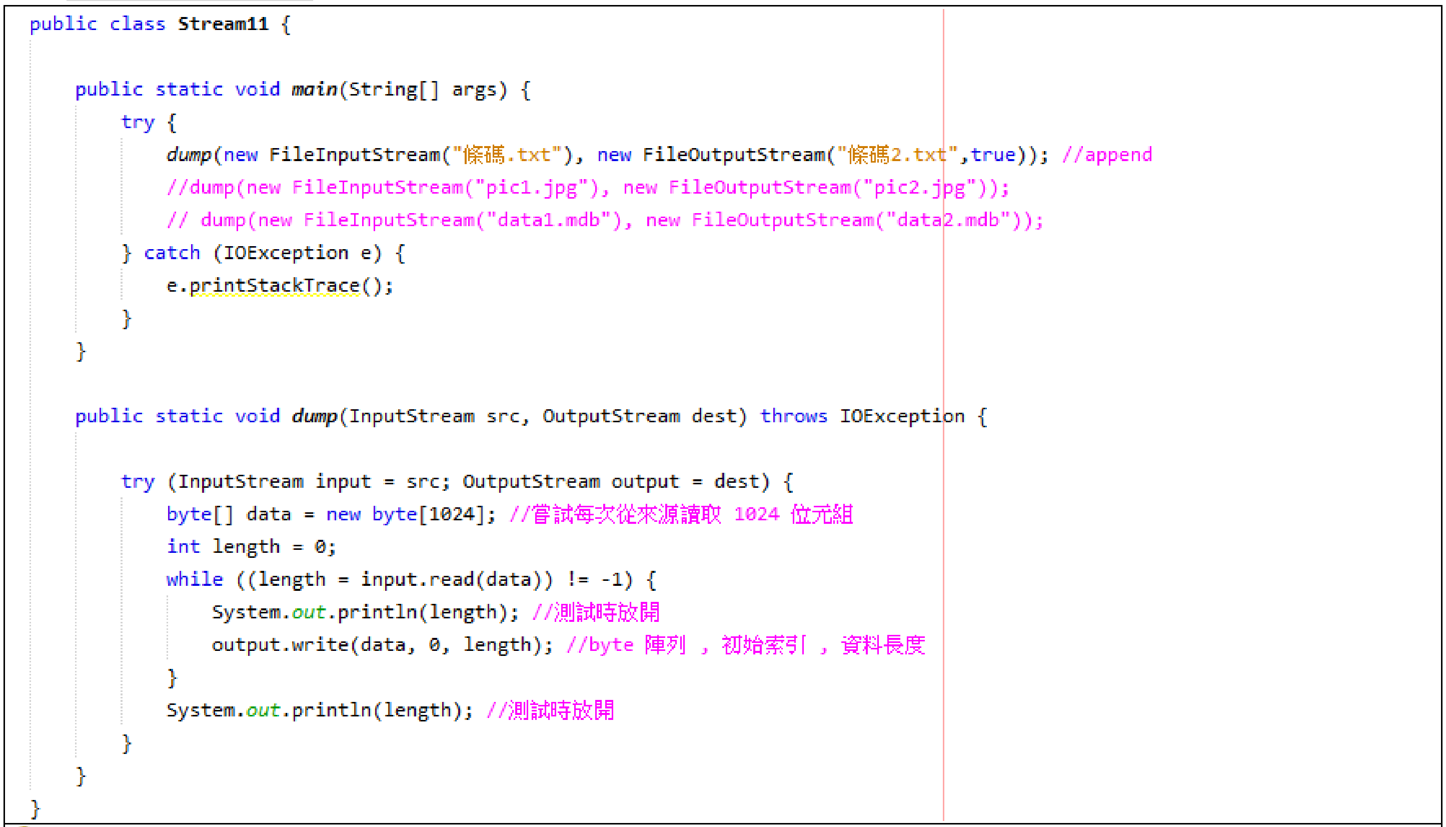
```
[c:\windows\addins]
[c:\windows\appcompat]
[c:\windows\AppPatch]
[c:\windows\AppReadiness]
c:\windows\Ascd_ProcessLog.ini
c:\windows\Ascd_tmp.ini
```



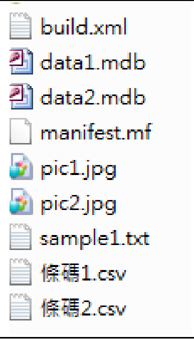
```java
public class File61 {
    //判斷是目錄還是檔案
    public static void main(String[] args) {
        try {
            File dir1 = new File("dir1");
            dir1.mkdir();

            File file1 = new File(dir1, "File1.txt");
            file1.createNewFile();

            System.out.println("dir1 is 目錄嗎 : " + dir1.isDirectory());
            System.out.println("file is 檔案嗎 : " + file1.isFile());
            System.out.println("file 存在嗎 : " + file1.exists());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

位元串流(I/O)➔相關類別

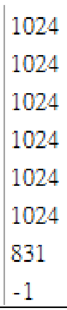
1. 如果要將資料從來源取出，可以使用輸入串流，如果要將資料寫入目的地，可以使用輸出串流
2. 無論資料來源或目的地為何，只要設法取得 InputStream 或 OutputStream ，接下來操作輸出輸入都是一致的，無需理會來源或目的地的真正形式
3. 不使用 InputStream 與 OutputStream 時，必須使用 close() 方法關閉串流，InputStream 與 OutputStream 因實作了 java.io.Closeable 介面，因此可使用 JDK7 嘗試自動關閉資源語法



```java
public class Stream11 {

    public static void main(String[] args) {
        try {
            dump(new FileInputStream("條碼.txt"), new FileOutputStream("條碼2.txt",true)); //append
            //dump(new FileInputStream("pic1.jpg"), new FileOutputStream("pic2.jpg"));
            // dump(new FileInputStream("data1.mdb"), new FileOutputStream("data2.mdb"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void dump(InputStream src, OutputStream dest) throws IOException {

        try (InputStream input = src; OutputStream output = dest) {
            byte[] data = new byte[1024]; //嘗試每次從來源讀取 1024 位元組
            int length = 0;
            while ((length = input.read(data)) != -1) {
                System.out.println(length); //測試時放開
                output.write(data, 0, length); //byte 陣列 , 初始索引 , 資料長度
            }
            System.out.println(length); //測試時放開
        }
    }
}
```



```
build.xml
data1.mdb
data2.mdb
manifest.mf
pic1.jpg
pic2.jpg
sample1.txt
條碼1.csv
條碼2.csv
```



```
1024
1024
1024
1024
1024
1024
831
-1
```

```
public class Stream12 {

    public static void main(String[] args) {
        String str = readFile12("條碼.txt");
        System.out.println(str);
    }
//在讀檔的過程中自行處理例外的發生➔不符合需求
    public static String readFile12(String name) {
        StringBuilder builder = new StringBuilder();
        try {
            InputStream input = new FileInputStream(name);
            Scanner scanner = new Scanner(input);
            while (scanner.hasNext()) {
                builder.append(scanner.nextLine());
                builder.append("\n");
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return builder.toString();
    }
}
```

```
public class Stream13 {
   //拋出給 呼叫者 main() 處理
    public static void main(String[] args) {
          try {
            String str = readFile13("條碼.txt");
            System.out.println(str);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
 public static String readFile13(String name) throws FileNotFoundException {

        StringBuilder builder = new StringBuilder();
        InputStream input = new FileInputStream(name);
        Scanner scanner = new Scanner(input);
        while (scanner.hasNext()) {
            builder.append(scanner.nextLine());
            builder.append('\n');
        }
        scanner.close(); //此行之前 若發生例外 , scanner.close()就不會做
        return builder.toString();
    }
}
```

如果最後一定要執行關閉資源的動作➔可寫在 finally 中，最後一定會執行到

但因 scanner 原先是 null，如果 FileInputStream 建構失敗，則 scanner 還是等於 null➔會拋出 NullPointerException

```java
public class Stream14 {

    public static void main(String[] args) {
        try {
            String str = readFile14("條碼.txt");
            System.out.println(str);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static String readFile14(String name) throws FileNotFoundException {
        StringBuilder builder = new StringBuilder();
        InputStream input = new FileInputStream(name); //假如失敗 , 會當在這一行
        Scanner scanner = new Scanner(input); //當上一行當掉 , 此行就不會做 , scanner 就會是 null
        try {
            while (scanner.hasNext()) {
                builder.append(scanner.nextLine());
                builder.append('\n');
            }
        } finally {
            scanner.close(); //如果 InputStream 建構失敗 , 此行會丟 NullPointerException
            return builder.toString();
        }
    }
}
```

JDK7 新增了嘗試關閉資源 try-with-resources

```java
public class Stream15 {

    public static void main(String[] args) {
          String str = "";
        try {
            str = readFile15("條碼.txt");
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        }
        System.out.println(str);
    }
  public static String readFile15(String name) throws FileNotFoundException {
        InputStream input = new FileInputStream(name);

        try (Scanner scanner = new Scanner(input)) {//自動嘗試關閉資源
            StringBuilder builder = new StringBuilder();

            while (scanner.hasNext()) {
                builder.append(scanner.nextLine());
                builder.append("\n");
            }
            return builder.toString();
        }
    }
}
```

反組譯時

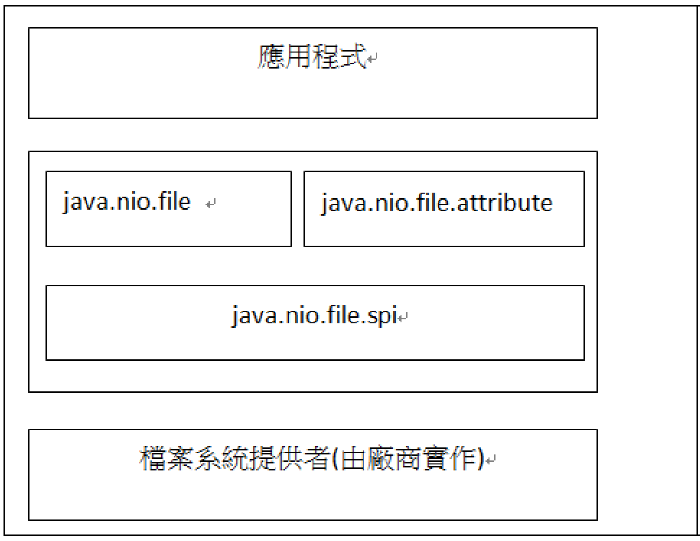
```java
public class Stream16 {

    public static void main(String[] args) {
        try {
            String str = readFile16("條碼.txt");
            System.out.println(str);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static String readFile16(String name) throws FileNotFoundException {
        StringBuilder builder = new StringBuilder();
        InputStream input = new FileInputStream(name);
        Scanner scanner = new Scanner(input);
        Throwable localThrowable2 = null;
        try {
            while (scanner.hasNext()) {
                builder.append(scanner.nextLine());
                builder.append('\n');
            }
        } catch (Throwable localThrowable1) {
            localThrowable2 = localThrowable1;
            throw localThrowable1;
        } finally {
            if (scanner != null) {
                try {
                    scanner.close();
                } catch (Throwable x2) {
                    localThrowable2.addSuppressed(x2); //可將第2個例外記錄在第1個例外中
                }
            } else {
                scanner.close();
            }
        }
        return builder.toString();
    }
}
```

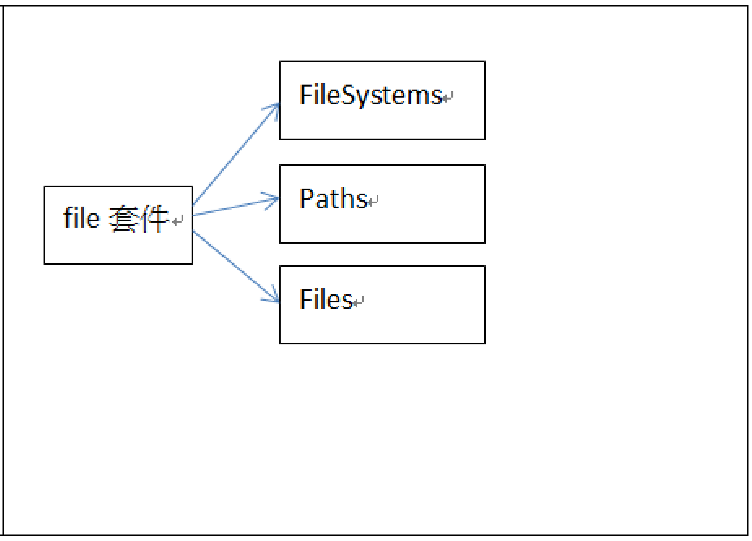
NIO2 檔案系統

1. Java SE 在 1.4 時加入了 NIO (New I/O) 的新 API，事隔多年後，在 Java SE 7 裡加入了第二代 NIO - NIO2
2. JDK6 之前常要針對特定檔案系統撰寫特定程式，不僅撰寫方式沒有標準，針對特定功能撰寫程式也會增加應用程式開發者的負擔

NIO2➔API 架構

1. NIO2 檔案系統 API 提供一組標準介面與類別，應用程式開發者只要基於這些標準介面與類別進行檔案系統操作，底層如何實作檔案系統操作，是由檔案系統提供者負責 (由廠商實作)
2. NIO2 檔案系統的中心是 java.nio.file.spi.FileSystemProvider，本身為抽象類別，是檔案系統提供者才要實作的類別，作用是產生 **java.nio.file** 與 **java.nio.file.attribute** 套件中各種抽象類別或介面的實作物件
3. 應用程式開發者使用 **java.nio.file** 與 **java.nio.file.attribute** 套件中必須實作的抽象類別或介面 (由檔案系統提供者實作)，應用程式開發者無需擔心底層實際如何存取檔案系統，只有檔案系統提供者才需關心 java.nio.file.spi 套件
4. 例如➔取得 java.nio.file.FileSystem
   FileSystem fileSystem= FileSystems.getDefault() ➔內部會使用 FileSystemProvider 實作物件的 getFileSystem()方法取得預設的 FileSystem 實作物件





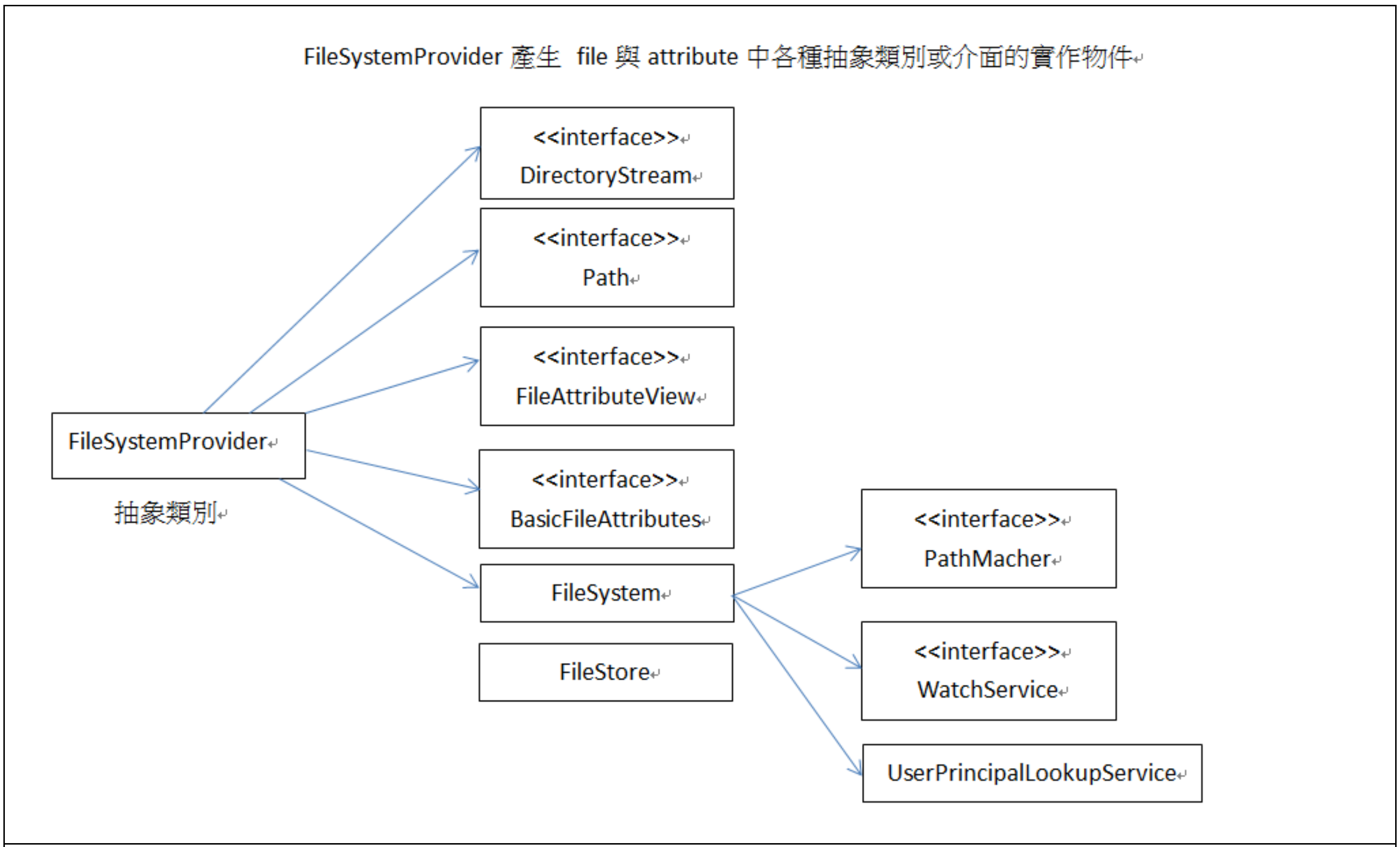
DirectoryStream➔指定路徑下的所有檔案(檔案與目錄名稱 可過濾)➔ Files.newDirectoryStream()

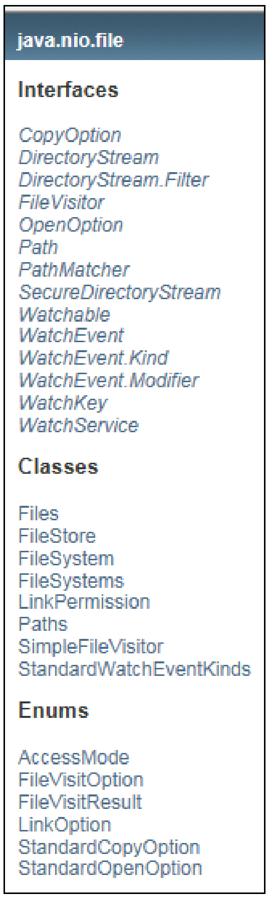
Path➔定義檔案 (由此開始)➔Paths.get()

BasicFileAttributes➔檔案屬性(建檔時間，修改時間)➔ Files.readAttributes()

FileSystem➔磁碟名稱，監聽目錄 ➔FileSystems.getDefault()，fs.getRootDirectories()，fs.newWatchService()

FileStore➔磁碟空間 (取得儲存體本身的資訊，位元組)➔Files.getFileStore() 或 fs.getFileStories()


FileSystemProvider 產生 file 與 attribute 中各種抽象類別或介面的實作物件
FileSystemProvider
抽象類別
<<interface>> DirectoryStream
<<interface>> Path
<<interface>> FileAttributeView
<<interface>> BasicFileAttributes
FileSystem
FileStore
<<interface>> PathMacher
<<interface>> WatchService
UserPrincipalLookupService


java.nio.file

Interfaces

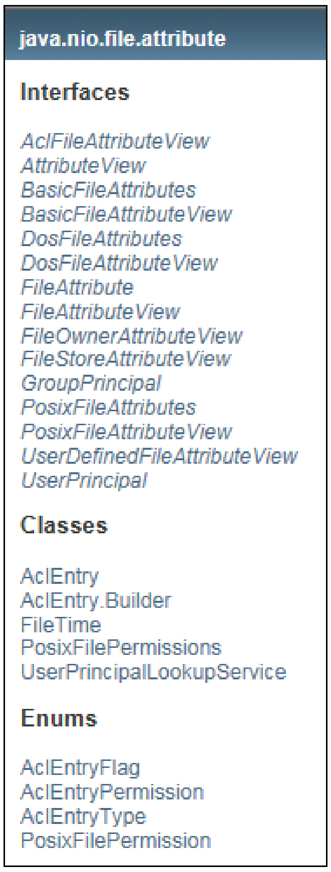
CopyOption
DirectoryStream
DirectoryStream.Filter
FileVisitor
OpenOption
Path
PathMatcher
SecureDirectoryStream
Watchable
WatchEvent
WatchEvent.Kind
WatchEvent.Modifier
WatchKey
WatchService

Classes

Files
FileStore
FileSystem
FileSystems
LinkPermission
Paths
SimpleFileVisitor
StandardWatchEventKinds

Enums

AccessMode
FileVisitOption
FileVisitResult
LinkOption
StandardCopyOption
StandardOpenOption


java.nio.file.attribute

Interfaces

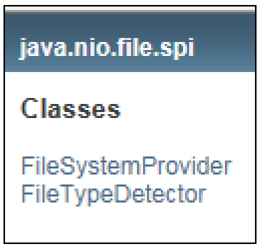
AclFileAttributeView
AttributeView
BasicFileAttributes
BasicFileAttributeView
DosFileAttributes
DosFileAttributeView
FileAttribute
FileAttributeView
FileOwnerAttributeView
FileStoreAttributeView
GroupPrincipal
PosixFileAttributes
PosixFileAttributeView
UserDefinedFileAttributeView
UserPrincipal

Classes

AclEntry
AclEntry.Builder
FileTime
PosixFilePermissions
UserPrincipalLookupService

Enums

AclEntryFlag
AclEntryPermission
AclEntryType
PosixFilePermission


java.nio.file.spi

Classes

FileSystemProvider
FileTypeDetector

http://docs.oracle.com/javase/7/docs/api/

路徑➔Path

1. 想要操作檔案，就得先指出檔案路徑，Path 實例 是在 JVM 中路徑的代表物件，也是 NIO2 檔案系統 API 操作的起點
2. 取得 Path 實例 ➔Paths.get()

```java
public class PathDemo1 {

    public static void main(String[] args) {
        Path path = Paths.get(System.getProperty("user.home"), "Documents", "Downloads");
        System.out.printf("toString: %s\n", path.toString());
        System.out.printf("getFileName: %s\n", path.getFileName());
        System.out.printf("getName(0): %s\n", path.getName(0)); //以資料夾為單位 ，最上層是 0
        System.out.printf("getNameCount: %d\n", path.getNameCount());
        System.out.printf("subpath(0,2): %s\n", path.subpath(0, 2));
        System.out.printf("getParent: %s\n", path.getParent());
        System.out.printf("getRoot: %s\n", path.getRoot());
    }
}
```

```
toString: C:\Users\joyes\Documents\Downloads
getFileName: Downloads
getName(0): Users
getNameCount: 4
subpath(0,2): Users\joyes
getParent: C:\Users\joyes\Documents
getRoot: C:\
```

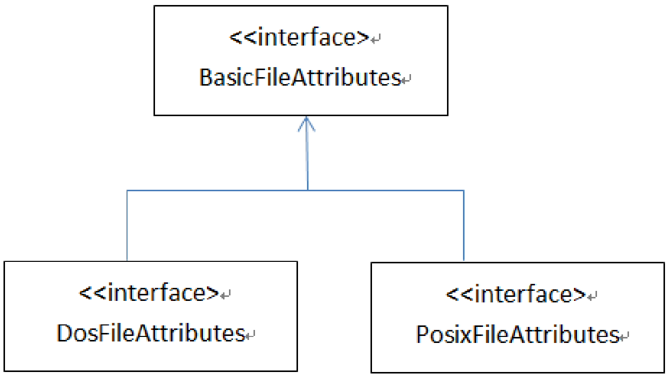
```java
//使用快捷迴圈
public class PathDemo2 {

    public static void main(String[] args) {
        Path path = Paths.get(System.getProperty("user.home"), "Documents", "Downloads");
        System.out.printf("toString: %s\n", path.toString());
        for (Path p : path) {
            System.out.println(p);
        }
    }
}
```

```
toString: C:\Users\joyes\Documents\Downloads
Users
joyes
Documents
Downloads
```

```
public class PathDemo3 {

    public static void main(String[] args) {
        //路徑與路徑的結合
        Path path1 = Paths.get("C:\\Users");
        Path path2 = Paths.get("joyes");
        Path path3 = path1.resolve(path2);
        System.out.printf("path3: %s\n", path3.toString());

        //從一個路徑切到另一個路徑
        Path path4 = Paths.get(System.getProperty("user.home"),"Documents", "Downloads");
        System.out.printf("path4: %s\n", path4.toString());
        Path path5 = Paths.get("C:\\Program files");
         System.out.printf("path5: %s\n", path5.toString());
        Path p4TOp5 = path4.relativize(path5);
        System.out.printf("p4TOp5: %s\n", p4TOp5.toString());
    }
}
```

```
path3: C:\Users\joyes
path4: C:\Users\joyes\Documents\Downloads
path5: C:\Program files
p4TOp5: ..\..\..\..\Program files
```

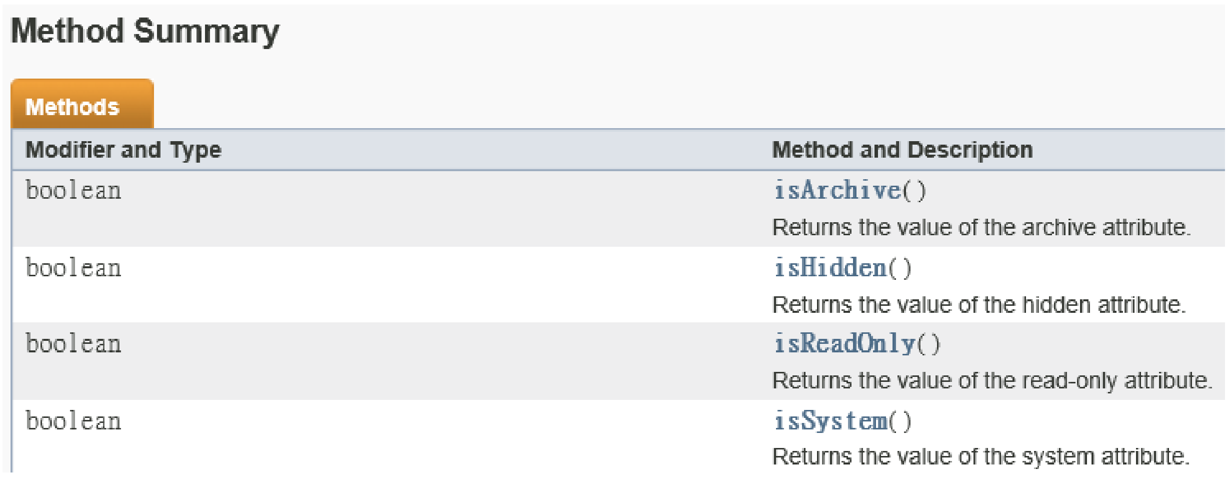
檔案屬性 ➔BaseFileAttributes (取得各檔案系統中都支援的屬性)、DosFileAttributes、PosixFileAttributes



1. 取得檔案屬性 ➔Files.readAttributes() 取得 BaseFileAttributes 物件

Path file = ...

DosFileAttributes attrs = Files.readAttributes(file, DosFileAttributes.class);



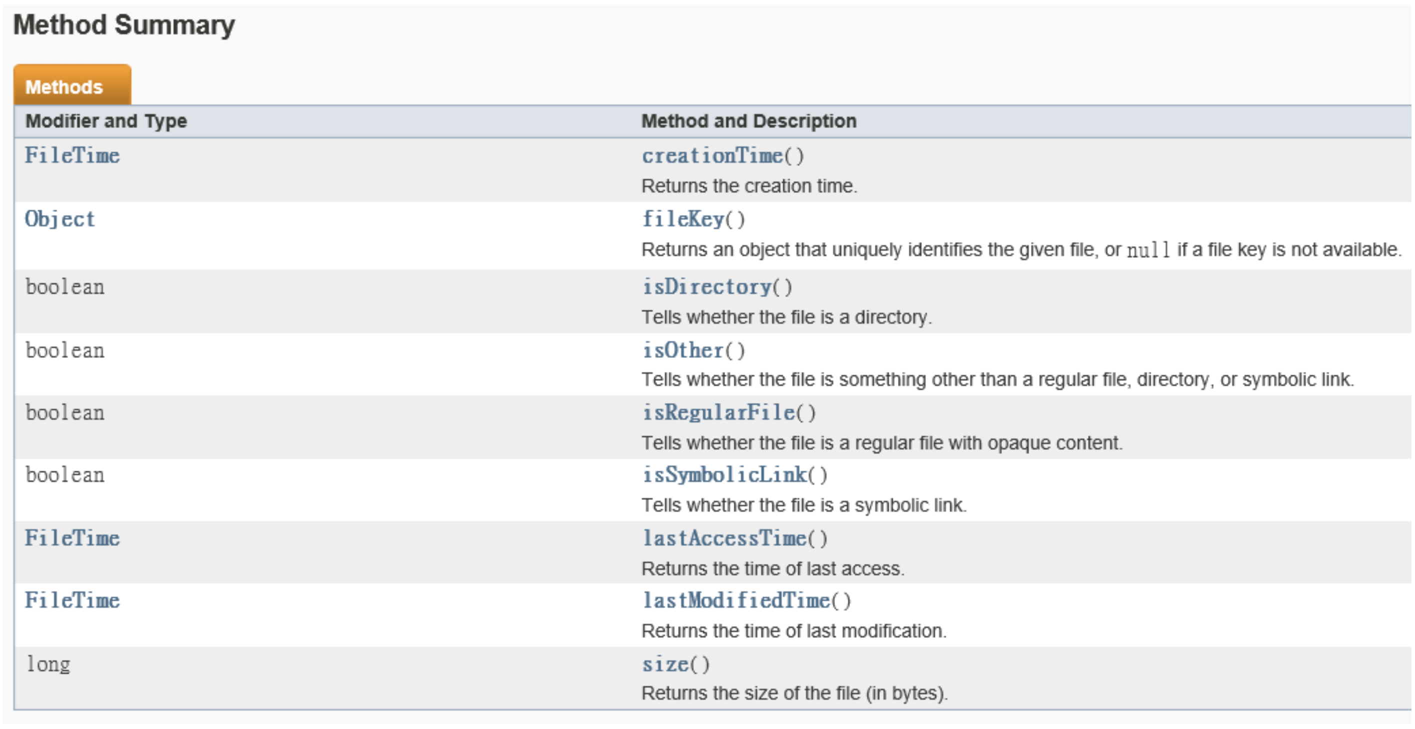
Method Summary

Methods

Modifier and Type	Method and Description
boolean	isArchive() Returns the value of the archive attribute.
boolean	isHidden() Returns the value of the hidden attribute.
boolean	isReadOnly() Returns the value of the read-only attribute.
boolean	isSystem() Returns the value of the system attribute.

Path file = ...

BasicFileAttributes attrs = Files.readAttributes(file, BasicFileAttributes.class);



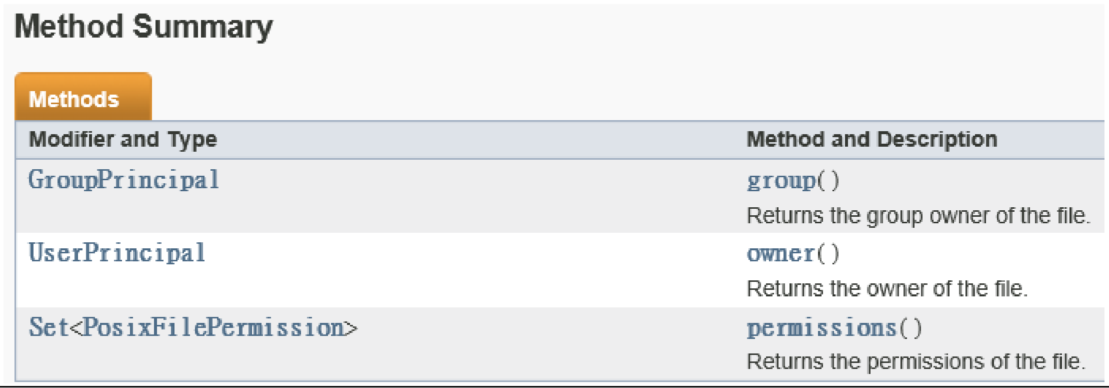
Method Summary

Methods

Modifier and Type	Method and Description
FileTime	creationTime() Returns the creation time.
Object	fileKey() Returns an object that uniquely identifies the given file, or null if a file key is not available.
boolean	isDirectory() Tells whether the file is a directory.
boolean	isOther() Tells whether the file is something other than a regular file, directory, or symbolic link.
boolean	isRegularFile() Tells whether the file is a regular file with opaque content.
boolean	isSymbolicLink() Tells whether the file is a symbolic link.
FileTime	lastAccessTime() Returns the time of last access.
FileTime	lastModifiedTime() Returns the time of last modification.
long	size() Returns the size of the file (in bytes).

Path file = ...

PosixFileAttributes attrs = Files.readAttributes(file, PosixFileAttributes.class);



Method Summary

Methods

Modifier and Type	Method and Description
GroupPrincipal	group() Returns the group owner of the file.
UserPrincipal	owner() Returns the owner of the file.
Set<PosixFilePermission>	permissions() Returns the permissions of the file.

```java
public class BasicFileAttributesDemo1 {

    public static void main(String[] args) throws IOException {

        Path file = Paths.get("data1.mdb");
        BasicFileAttributes attrs = Files.readAttributes(file, BasicFileAttributes.class);
        System.out.printf("creationTime: %s\n", attrs.creationTime());
        System.out.printf("lastAccessTime: %s\n", attrs.lastAccessTime());
        System.out.printf("lastModifiedTime: %s\n", attrs.lastModifiedTime());
        System.out.printf("isDirectory: %b\n", attrs.isDirectory());

        //捷徑
        System.out.printf("isSymbolicLink: %b\n", attrs.isSymbolicLink());
        System.out.printf("size: %d\n", attrs.size());
    }
}
```

```
creationTime: 2016-06-08T01:49:32.365066Z
lastAccessTime: 2016-06-08T01:49:32.365066Z
lastModifiedTime: 2013-05-21T04:54:40Z
isDirectory: false
isSymbolicLink: false
size: 720896
```

2. 設定檔案屬性➔Files.setAttribute()

<1>、 Files.getLastModifiedTime()，Files.setLastModifiedTime() 只是簡便的方法

<2>、 可透過 Files.setAttribute()方法

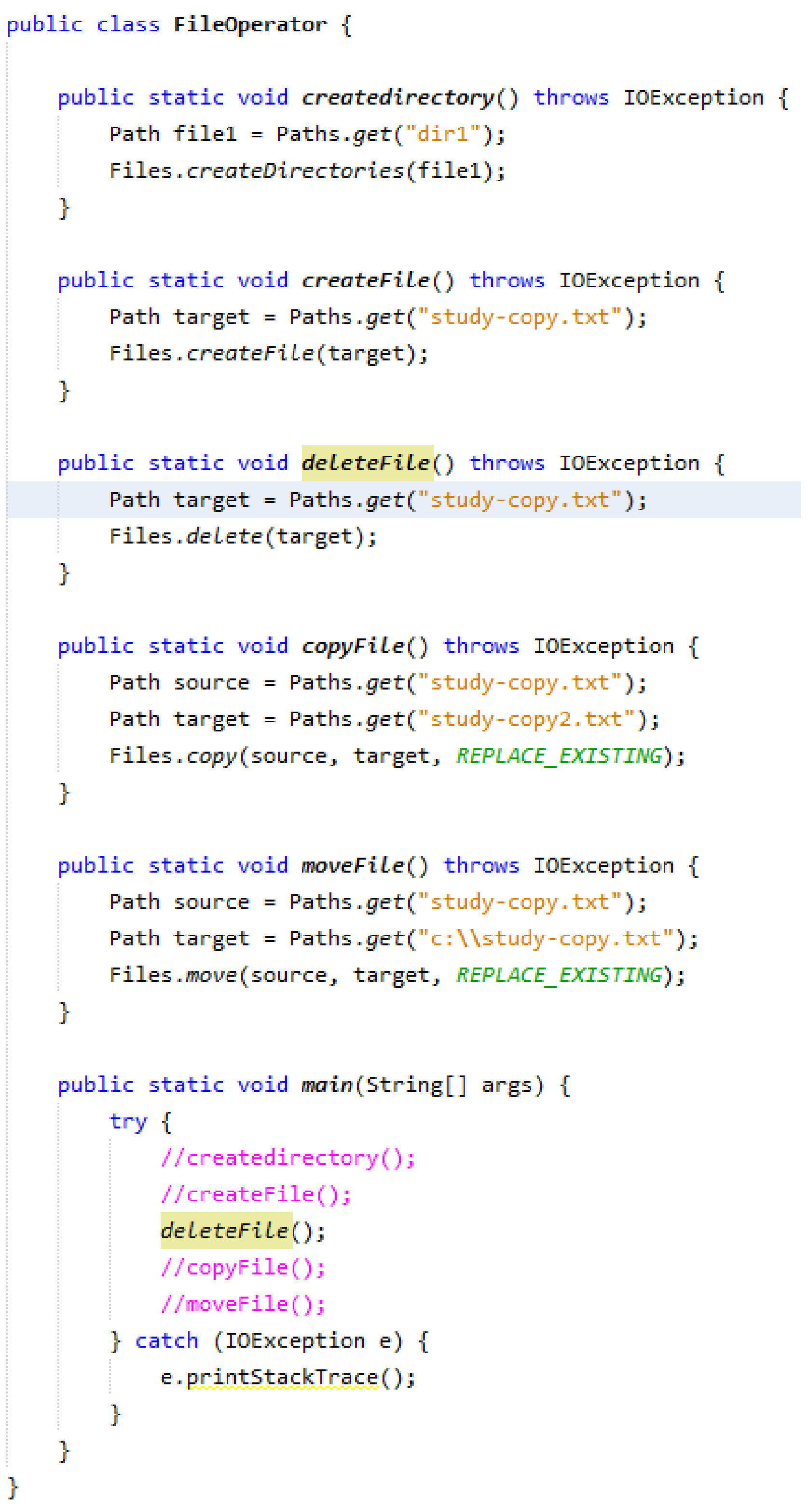
```java
public class BasicFileAttributesDemo2 {

    public static void main(String[] args) throws IOException {
        Path file = Paths.get("abc.txt");
        //設定最後修改時間
        long currentTime = System.currentTimeMillis();
        FileTime ft = FileTime.fromMillis((currentTime));

        Files.setLastModifiedTime(file, ft);
    }
}
```

```
public class BasicFileAttributesDemo3 {

    public static void main(String[] args) throws IOException {
        Path file = Paths.get("abc.txt");
        //設定最後修改時間
        long currentTime = System.currentTimeMillis();
        FileTime ft = FileTime.fromMillis((currentTime));

        // [viewname:]attribute-name
        Files.setAttribute(file, "basic:lastModifiedTime", ft);
        //隱藏屬性
        Files.setAttribute(file, "dos:hidden", false);
    }
}
```

操作檔案與目錄

1. Files.createFile()
2. Files.createDirectory()
3. Files.delete()、Files.deleteIfExists()
4. Files.copy()➔第 3 個參數 指定 CopyOption 介面的實作物件
5. Files.move()

```
public enum StandardCopyOption implements CopyOption {
    /**
     * Replace an existing file if it exists.
     */
    REPLACE_EXISTING,
    /**
     * Copy attributes to the new file.
     */
    COPY_ATTRIBUTES,
    /**
     * Move the file as an atomic file system operation.
     */
    ATOMIC_MOVE;
}
```



```java
public class FileOperator {

    public static void createdirectory() throws IOException {
        Path file1 = Paths.get("dir1");
        Files.createDirectories(file1);
    }

    public static void createFile() throws IOException {
        Path target = Paths.get("study-copy.txt");
        Files.createFile(target);
    }

    public static void deleteFile() throws IOException {
        Path target = Paths.get("study-copy.txt");
        Files.delete(target);
    }

    public static void copyFile() throws IOException {
        Path source = Paths.get("study-copy.txt");
        Path target = Paths.get("study-copy2.txt");
        Files.copy(source, target, REPLACE_EXISTING);
    }

    public static void moveFile() throws IOException {
        Path source = Paths.get("study-copy.txt");
        Path target = Paths.get("c:\\study-copy.txt");
        Files.move(source, target, REPLACE_EXISTING);
    }

    public static void main(String[] args) {
        try {
            //createdirectory();
            //createFile();
            deleteFile();
            //copyFile();
            //moveFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

指定路徑下的所有檔案➔DirectoryStream

1. 使用 Files.newDirectoryStream()方法取得 DirectoryStream 介面實作物件，代表指定路徑下的所有檔案➔只列出在目錄內的檔案不會深入到目錄內，因繼承了 Closeable 介面，父介面為 AutoCloseable 可搭配嘗試關閉資源語法

```java
public class DirectoryStreamDir {

//會先列出目錄 ，再列出檔案
    public static void main(String[] args) throws IOException {

        Path path = Paths.get("c:\\");
        try (DirectoryStream<Path> stream = Files.newDirectoryStream(path)) {
            List<String> files = new ArrayList<>();

            for (Path path1 : stream) {

                if (Files.isDirectory(path1)) {
                    System.out.printf("[%s]\n", path1.getFileName()); //目錄直接印
                } else {
                    files.add(path1.getFileName().toString()); //把檔案加進集合內
                }
            }
            for (String file : files) { //最後再把檔案印出來
                System.out.println(file);
            }
        }
    }
}
```

```
[$Recycle.Bin]
[Dev-Cpp]
[Documents and Settings]
[Kpcms]
[MSOCache]
[PerfLogs]
[Program Files]
[ProgramData]
[Recovery]
[System Volume Information]
[Users]
[Windows]
autoexec.bat
config.sys
desktop.ini
hiberfil.sys
pagefile.sys
xyz.txt
```

2. 過濾搜尋檔案➔glob

<1>、 使用 Files.newDirectoryStream() 時的第二個參數指定過濾條件

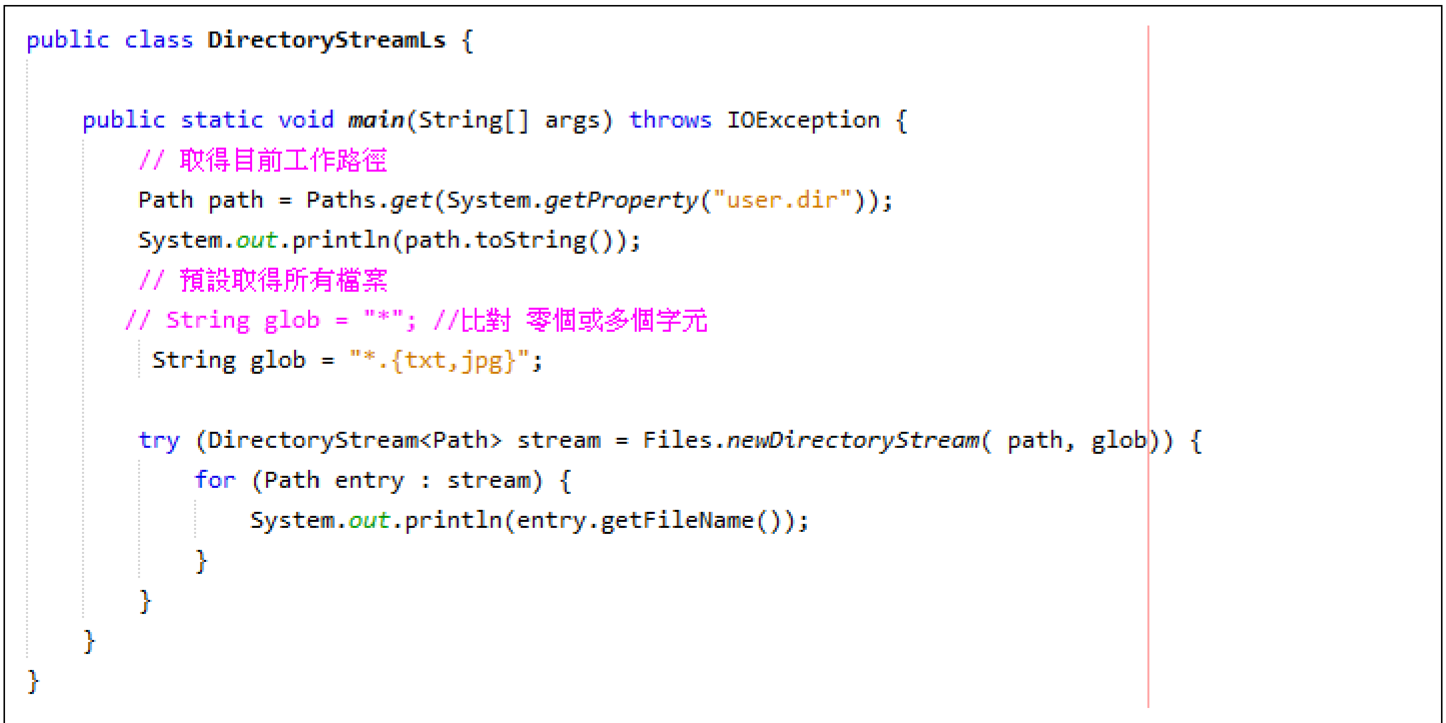
*	比對零個或多個字元
?	比對一個字元
{ }	比對收集的任一子模式，例如{class，jar } 比對 class 或 jar {tmp,temp*} 比對 tmp 或 temp 開頭
[ ]	比對收集的任一字元 例如 [a-z] 比對 a 到 z 任一字元
\	忽略符號 要比對 *，?，\ 就要寫成 *，\?，\\
其它字元	比對字元本身

<2>、 *.{class,jar} 這樣的語法稱之為 Glob，是一種模式比對語法，常用於目錄與檔案的比對

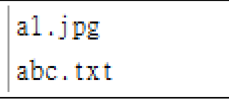
(1). *.java➔比對 .java 結尾的字串

(2). ??? 符合三個字元 ➔123，abc 會符合

(3). *.{class，jar }➔符合 .class 或 .jar 結尾的字串

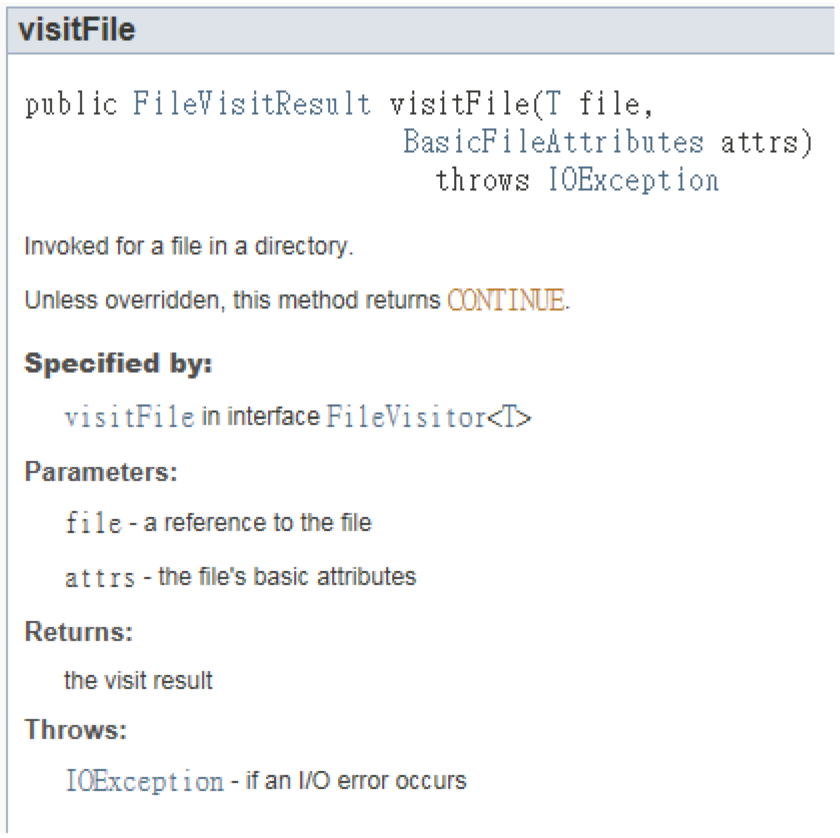


```java
public class DirectoryStreamLs {

    public static void main(String[] args) throws IOException {
        // 取得目前工作路徑
        Path path = Paths.get(System.getProperty("user.dir"));
        System.out.println(path.toString());
        // 預設取得所有檔案
       // String glob = "*"; //比對 零個或多個字元
         String glob = "*.{txt,jpg}";

        try (DirectoryStream<Path> stream = Files.newDirectoryStream( path, glob)) {
            for (Path entry : stream) {
                System.out.println(entry.getFileName());
            }
        }
    }
}
```



```
a1.jpg
abc.txt
```

3. 走訪目錄下的檔案➔ Files.walkFileTree()➔深入到子目錄內
   <1>、 可以實作 FileVisitor 介面，其中定義了 四個必須實作的方法，如果只對 FileVisitor 一，兩個方法有興趣，可以繼承 SimpleFileVisitor 類別，這個類別實作了 FileVisitor 介面，只要繼承之後重新定義感興趣的方法就可以了
   <2>、 從指定的目錄路徑開始，每次要開始走訪該目錄內容前，會呼叫 preVisitDirectory，要走訪檔案時會呼叫 visitFile，走訪檔案失敗會呼叫 visitFileFailed，走訪整個目錄內後會呼叫 postVisitDirectory()
   <3>、 如果要使用 FileVisitor 走訪目錄，可以使用 Files.walkFileTree()方法



visitFile

```
public FileVisitResult visitFile(T file,
                                 BasicFileAttributes attrs)
                          throws IOException
```

Invoked for a file in a directory.

Unless overridden, this method returns CONTINUE.

Specified by:

visitFile in interface FileVisitor<T>

Parameters:

file - a reference to the file

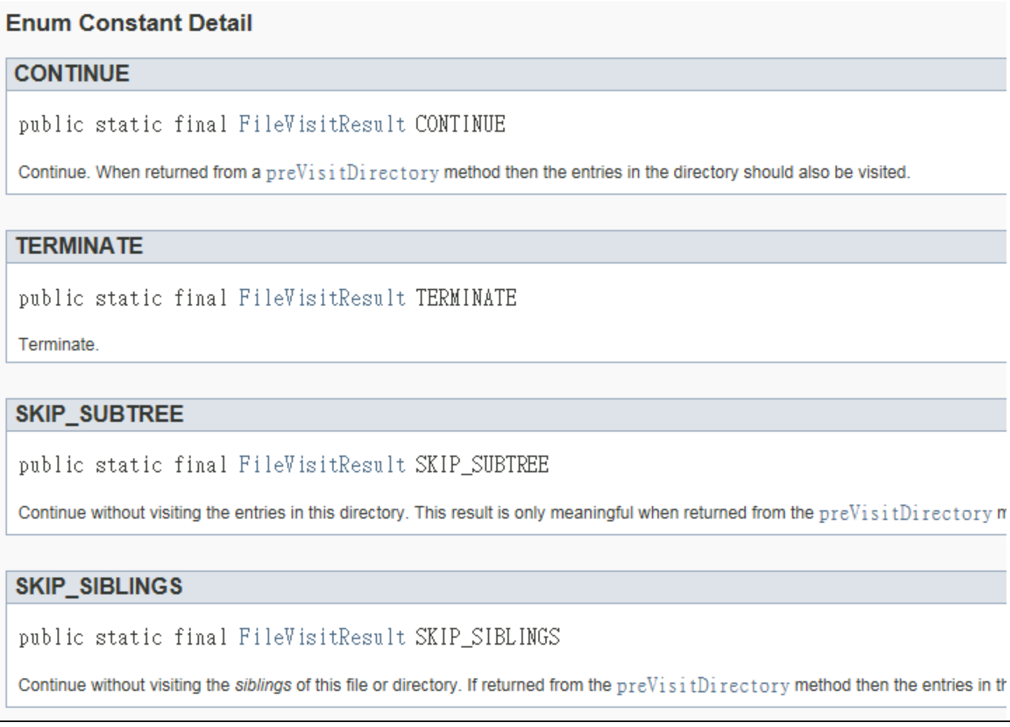
attrs - the file's basic attributes

Returns:

the visit result

Throws:

IOException - if an I/O error occurs



Enum Constant Detail

CONTINUE

```
public static final FileVisitResult CONTINUE
```

Continue. When returned from a preVisitDirectory method then the entries in the directory should also be visited.

TERMINATE

```
public static final FileVisitResult TERMINATE
```

Terminate.

SKIP_SUBTREE

```
public static final FileVisitResult SKIP_SUBTREE
```

Continue without visiting the entries in this directory. This result is only meaningful when returned from the preVisitDirectory m

SKIP_SIBLINGS

```
public static final FileVisitResult SKIP_SIBLINGS
```

Continue without visiting the *siblings* of this file or directory. If returned from the preVisitDirectory method then the entries in th

```
public class DirectoryStreamDirAll {

    public static void main(String[] args) throws IOException {
        Path path = Paths.get("c:\\windows");
        Files.walkFileTree(path, new ConsoleFileVisitor());
    }
}

class ConsoleFileVisitor extends SimpleFileVisitor<Path> {

    @Override
    public FileVisitResult visitFile(Path path, BasicFileAttributes attr) {
        printSpace(path);
        System.out.printf("%s\n", path.getFileName());
        return FileVisitResult.CONTINUE;
    }

    @Override
    public FileVisitResult preVisitDirectory(Path path, BasicFileAttributes attrs) throws IOException {
        printSpace(path);
        System.out.printf("[%s]\n", path.getFileName());
        return FileVisitResult.CONTINUE;
    }

    @Override
    public FileVisitResult visitFileFailed(Path file, IOException exc) {
        System.err.println(exc);
        return FileVisitResult.CONTINUE;
    }

    private void printSpace(Path path) {
        System.out.printf("%" + path.getNameCount() * 2 + "s", " ");
    }
}
```

```
[windows]
  [addins]
    FXSEXT.ecf
  [AppCompat]
    [Programs]
      AEINV_PREVIOUS.xml
      AEINV_WER_{EC733E1B-2436-48B8-97B0-8E3CA78100C8}_20120203_152630.xml
      RecentFileCache.bcf
```

IO 當中，有個 file.listFiles()可以回傳此 file 路徑下所有的檔案與資料夾，回傳的型態是 File 陣列，

然後根據此陣列長度做 for 迴圈的走訪，再用 isFile() 來判斷是不是檔案，

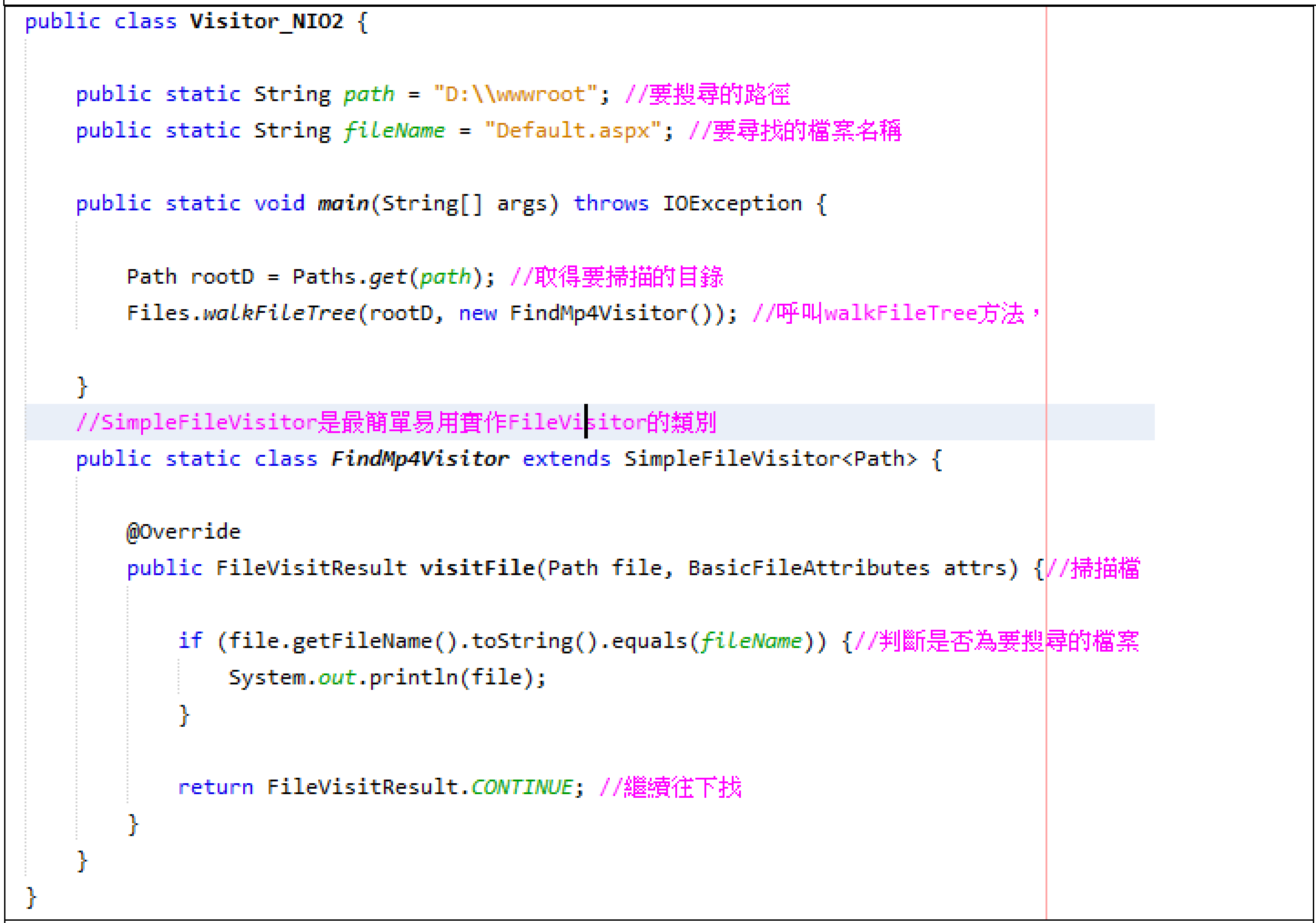
不是的話就表示是資料夾，是資料夾的話就用此 file 路徑再進入下一層重覆做上面的動作。

是檔案的話，就呼叫 getName()，取得檔名(String)，然後進行 equals 的字串比對，

將找到的檔名印出。因為要做重覆動作，所以這些都要寫在一個方法裡面

```java
public class Visitor_IO {

    public static String path = "D:\\wwwroot"; //要搜尋的路徑
    public static String fileName = "Default.aspx"; //要尋找的檔案名稱

    public static void main(String arg[]) {

        File file = new File(path);
        method(file);

    }

    public static void method(File file) {
        File f[] = file.listFiles(); //回傳file目錄下的所有檔案與資料夾

        for (int i = 0; i < f.length; i++) {

            if (f[i].isFile()) {//走訪所有目錄，如果不是檔案則進入下一層重覆此搜尋動作
                if (f[i].getName().equals(fileName)) {//是檔案的話，則判斷是不是要找尋的目標
                    System.out.println(f[i]);
                }
            } else {
                method(f[i]);
            }
        }
    }
}
```

```
D:\wwwroot\certsystem\Default.aspx
D:\wwwroot\docsystem\Default.aspx
D:\wwwroot\factorsystem\Default.aspx
D:\wwwroot\formsystem\Default.aspx
D:\wwwroot\inssystem\Default.aspx
D:\wwwroot\mobile\Default.aspx
D:\wwwroot\opersystem\Default.aspx
D:\wwwroot\pursystem\Default.aspx
D:\wwwroot\smartsystem\Default.aspx
D:\wwwroot\sscmsystem\Default.aspx
D:\wwwroot\techsystem\Default.aspx
```



```java
public class Visitor_NIO2 {

    public static String path = "D:\\wwwroot"; //要搜尋的路徑
    public static String fileName = "Default.aspx"; //要尋找的檔案名稱

    public static void main(String[] args) throws IOException {

        Path rootD = Paths.get(path); //取得要掃描的目錄
        Files.walkFileTree(rootD, new FindMp4Visitor()); //呼叫walkFileTree方法，

    }
    //SimpleFileVisitor是最簡單易用實作FileVisitor的類別
    public static class FindMp4Visitor extends SimpleFileVisitor<Path> {

        @Override
        public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) {//掃描檔

            if (file.getFileName().toString().equals(fileName)) {//判斷是否為要搜尋的檔案
                System.out.println(file);
            }

            return FileVisitResult.CONTINUE; //繼續往下找
        }
    }
}
```

```
D:\wwwroot\certsystem\Default.aspx
D:\wwwroot\docsystem\Default.aspx
D:\wwwroot\factorsystem\Default.aspx
D:\wwwroot\formsystem\Default.aspx
D:\wwwroot\inssystem\Default.aspx
D:\wwwroot\mobile\Default.aspx
D:\wwwroot\opersystem\Default.aspx
D:\wwwroot\pursystem\Default.aspx
D:\wwwroot\smartsystem\Default.aspx
D:\wwwroot\sscmsystem\Default.aspx
D:\wwwroot\techsystem\Default.aspx
```

讀取，走訪磁碟➔FileSystem

1. 使用 FileSystem 物件 的 getRootDirectories()方法，會取回 Interable<String>

```
public class FileSystemRoots {

    public static void main(String[] args) {

        FileSystem fs = FileSystems.getDefault();
        Iterable<Path> dirs =fs.getRootDirectories();

        for (Path name : dirs) {
            System.out.println(name);
        }
    }
}
```

```
A:\
C:\
D:\
E:\
G:\
I:\
J:\
X:\
```

2. Watching a Directory for Changes➔對於資料夾監聽是否有刪除修改新增

<1>、 先建立一個你要監聽的對象 例如我想監聽 src 資料夾

Path path = Paths.get("..\\src");

<2>、 建立一個 WatchService，可由 **FileSystems.getDefault().newWatchService()**

<3>、 取得建立時記得要 try catch

將 WatchService 與需監聽的狀態 註冊到 Path 上 程式碼如下

因為 Path 有實作 Watchable 介面才可註冊 WatchService

```
try {
watchService = FileSystems.getDefault().newWatchService();
path.register(watchService, StandardWatchEventKinds.ENTRY_MODIFY,
StandardWatchEventKinds.ENTRY_CREATE,
StandardWatchEventKinds.ENTRY_DELETE);
} catch (IOException e) {
e.printStackTrace();
}
```

<4>、 建立一個無窮迴圈他取得資料夾修改資訊 watchService 有三種方式可以取得修改資訊

(1). poll 會返回 WatchKey

(2). poll(timeout, unit) 可指定等待時間返回 WatchKey

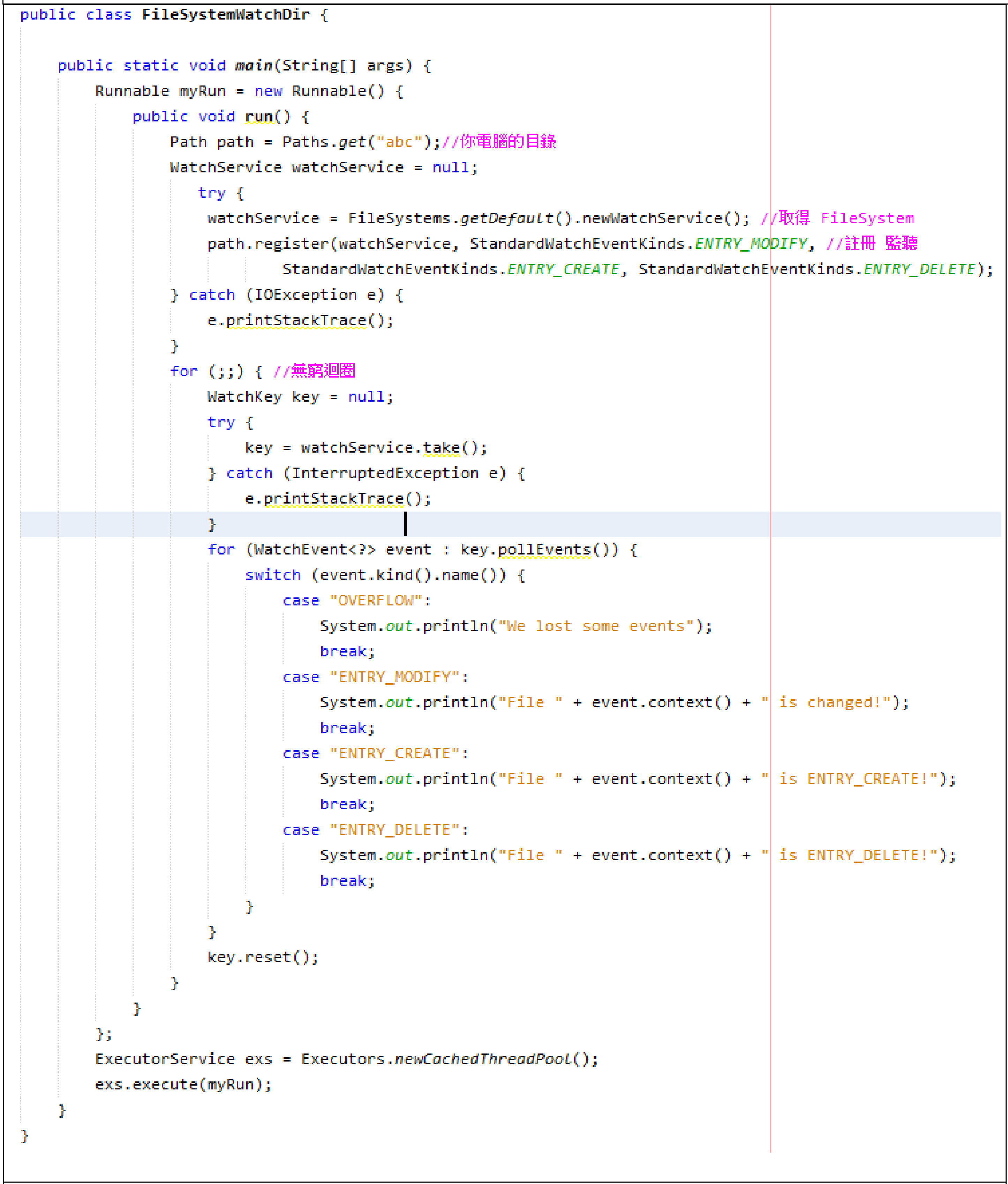
(3). take 如序列中沒資料時會 wait 也返回 WatchKey

<5>、 透過 WatchKey 當中的 **pollEvents** 使用 For 迴圈取得所有註冊狀態的回應

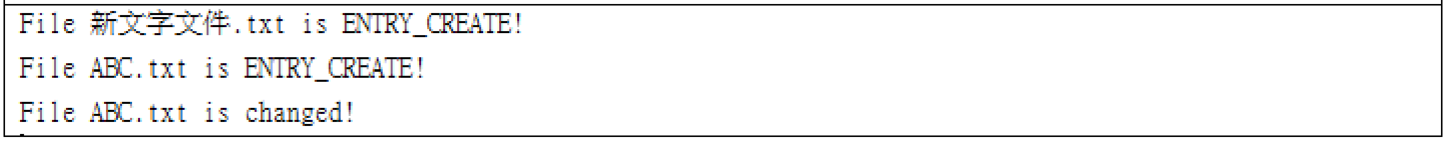
<6>、 最後記得需要呼叫 WatchKey 的 **reset** 將狀態變成 Ready,不然會收不到新的訊息

你的的程序中可能會收到一個 OVERFLOW 事件，甚至如果該程序被沒有針對此事件註冊。

如果你註冊在根目錄其子目錄並不會有監聽效果！



```
public class FileSystemWatchDir {

    public static void main(String[] args) {
        Runnable myRun = new Runnable() {
            public void run() {
                Path path = Paths.get("abc");//你電腦的目錄
                WatchService watchService = null;
                   try {
                    watchService = FileSystems.getDefault().newWatchService(); //取得 FileSystem
                    path.register(watchService, StandardWatchEventKinds.ENTRY_MODIFY, //註冊 監聽
                            StandardWatchEventKinds.ENTRY_CREATE, StandardWatchEventKinds.ENTRY_DELETE);
                } catch (IOException e) {
                    e.printStackTrace();
                }
                for (;;) { //無窮迴圈
                    WatchKey key = null;
                    try {
                        key = watchService.take();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    for (WatchEvent<?> event : key.pollEvents()) {
                        switch (event.kind().name()) {
                            case "OVERFLOW":
                                System.out.println("We lost some events");
                                break;
                            case "ENTRY_MODIFY":
                                System.out.println("File " + event.context() + " is changed!");
                                break;
                            case "ENTRY_CREATE":
                                System.out.println("File " + event.context() + " is ENTRY_CREATE!");
                                break;
                            case "ENTRY_DELETE":
                                System.out.println("File " + event.context() + " is ENTRY_DELETE!");
                                break;
                        }
                    }
                    key.reset();
                }
            }
        };
        ExecutorService exs = Executors.newCachedThreadPool();
        exs.execute(myRun);
    }
}
```



```
File 新文字文件.txt is ENTRY_CREATE!
File ABC.txt is ENTRY_CREATE!
File ABC.txt is changed!
```

儲存體本身的資訊 ➔FileStore

1. FileSystems.getDefault() 取的 FileSystem 物件 ➔ FileSystem(物件) .getFileStores() 取得 FileStore 物件
2. Files.getFileStore() 取得 FileStore 物件

```
public class FileStoreDisk {

    public static void main(String[] args) throws IOException {
        Path path = Paths.get("c:\\");

//全部磁碟
        FileSystem fs = FileSystems.getDefault();
        for (FileStore store : fs.getFileStores()) {
            printDisk(store);
        }
//指定單一磁碟
        FileStore store = Files.getFileStore(path);
        printDisk(store);
    }

    public static void printDisk(FileStore store) throws IOException {
        long total = store.getTotalSpace();
        long used = store.getTotalSpace() - store.getUnallocatedSpace();
        long usable = store.getUsableSpace();
        DecimalFormat formatter = new DecimalFormat("#,###,###");
        System.out.println(store.toString());
        System.out.printf("\t- 總容量\t%s\t位元組\n", formatter.format(total));
        System.out.printf("\t- 已用空間\t%s\t位元組\n", formatter.format(used));
        System.out.printf("\t- 可用空間\t%s\t位元組\n", formatter.format(usable));
    }
}
```

```
(C:)
        - 總容量        104,857,595,904 位元組
        - 已用空間      45,335,470,080  位元組
        - 可用空間      59,522,125,824  位元組
新增磁碟區 (D:)
        - 總容量        262,143,995,904 位元組
        - 已用空間      26,750,431,232  位元組
        - 可用空間      235,393,564,672 位元組
Removable Disk (F:)
        - 總容量        16,173,203,456  位元組
        - 已用空間      1,012,363,264   位元組
        - 可用空間      15,160,840,192  位元組
新增磁碟區 (J:)
        - 總容量        132,998,230,016 位元組
        - 已用空間      121,245,696     位元組
        - 可用空間      132,876,984,320 位元組
(c:)
        - 總容量        104,857,595,904 位元組
        - 已用空間      45,335,474,176  位元組
        - 可用空間      59,522,121,728  位元組
```

Paths

Modifier and Type	Method and Description
static Path	get(String first, String... more) Converts a path string, or a sequence of strings that when joined form a path string, to a Path.
static Path	get(URI uri) Converts the given URI to a Path object.

Path

Modifier and Type	Method and Description
int	compareTo(Path other) Compares two abstract paths lexicographically.
boolean	endsWith(Path other) Tests if this path ends with the given path.
boolean	endsWith(String other) Tests if this path ends with a Path, constructed by converting the given path string, in exactly the manner specified by the endsWith(Path) method.
boolean	equals(Object other) Tests this path for equality with the given object.
Path	getFileName() Returns the name of the file or directory denoted by this path as a Path object.
FileSystem	getFileSystem() Returns the file system that created this object.
Path	getName(int index) Returns a name element of this path as a Path object.
int	getNameCount() Returns the number of name elements in the path.
Path	getParent() Returns the parent path, or null if this path does not have a parent.
Path	getRoot() Returns the root component of this path as a Path object, or null if this path does not have a root component.
int	hashCode() Computes a hash code for this path.
boolean	isAbsolute() Tells whether or not this path is absolute.
Iterator<Path>	iterator() Returns an iterator over the name elements of this path.
Path	normalize() Returns a path that is this path with redundant name elements eliminated.
WatchKey	register(WatchService watcher, WatchEvent.Kind<?>... events) Registers the file located by this path with a watch service.
WatchKey	register(WatchService watcher, WatchEvent.Kind<?>[] events, WatchEvent.Modifier... modifiers) Registers the file located by this path with a watch service.

Path	relativize(Path other) Constructs a relative path between this path and a given path.
Path	resolve(Path other) Resolve the given path against this path.
Path	resolve(String other) Converts a given path string to a Path and resolves it against this Path in exactly the manner specified by the resolve method.
Path	resolveSibling(Path other) Resolves the given path against this path's parent path.
Path	resolveSibling(String other) Converts a given path string to a Path and resolves it against this path's parent path in exactly the manner specified by the resolveSibling method.
boolean	startsWith(Path other) Tests if this path starts with the given path.
boolean	startsWith(String other) Tests if this path starts with a Path, constructed by converting the given path string, in exactly the manner specified by the startsWith(Path) method.
Path	subpath(int beginIndex, int endIndex) Returns a relative Path that is a subsequence of the name elements of this path.
Path	toAbsolutePath() Returns a Path object representing the absolute path of this path.
File	toFile() Returns a File object representing this path.
Path	toRealPath(LinkOption... options) Returns the real path of an existing file.
String	toString() Returns the string representation of this path.
URI	toUri() Returns a URI to represent this path.

Files

Modifier and Type	Method and Description
static long	copy(InputStream in, Path target, CopyOption... options) Copies all bytes from an input stream to a file.
static long	copy(Path source, OutputStream out) Copies all bytes from a file to an output stream.
static Path	copy(Path source, Path target, CopyOption... options) Copy a file to a target file.
static Path	createDirectories(Path dir, FileAttribute<?>... attrs) Creates a directory by creating all nonexistent parent directories first.
static Path	createDirectory(Path dir, FileAttribute<?>... attrs) Creates a new directory.

static Path	createFile(Path path, FileAttribute<?>... attrs) Creates a new and empty file, failing if the file already exists.
static Path	createLink(Path link, Path existing) Creates a new link (directory entry) for an existing file (optional operation).
static Path	createSymbolicLink(Path link, Path target, FileAttribute<?>... attrs) Creates a symbolic link to a target (optional operation).
static Path	createTempDirectory(Path dir, String prefix, FileAttribute<?>... attrs) Creates a new directory in the specified directory, using the given prefix to generate its name.
static Path	createTempDirectory(String prefix, FileAttribute<?>... attrs) Creates a new directory in the default temporary-file directory, using the given prefix to generate its name.
static Path	createTempFile(Path dir, String prefix, String suffix, FileAttribute<?>... attrs) Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name.
static Path	createTempFile(String prefix, String suffix, FileAttribute<?>... attrs) Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
static void	delete(Path path) Deletes a file.
static boolean	deleteIfExists(Path path) Deletes a file if it exists.
static boolean	exists(Path path, LinkOption... options) Tests whether a file exists.
static Object	getAttribute(Path path, String attribute, LinkOption... options) Reads the value of a file attribute.
static <V extends FileAttributeView> V	getFileAttributeView(Path path, Class<V> type, LinkOption... options) Returns a file attribute view of a given type.
static FileStore	getFileStore(Path path) Returns the FileStore representing the file store where a file is located.
static FileTime	getLastModifiedTime(Path path, LinkOption... options) Returns a file's last modified time.
static UserPrincipal	getOwner(Path path, LinkOption... options) Returns the owner of a file.
static Set<PosixFilePermission>	getPosixFilePermissions(Path path, LinkOption... options) Returns a file's POSIX file permissions.
static boolean	isDirectory(Path path, LinkOption... options) Tests whether a file is a directory.
static boolean	isExecutable(Path path) Tests whether a file is executable.
static boolean	isHidden(Path path) Tells whether or not a file is considered hidden.

static boolean	isReadable(Path path) Tests whether a file is readable.
static boolean	isRegularFile(Path path, LinkOption... options) Tests whether a file is a regular file with opaque content.
static boolean	isSameFile(Path path, Path path2) Tests if two paths locate the same file.
static boolean	isSymbolicLink(Path path) Tests whether a file is a symbolic link.
static boolean	isWritable(Path path) Tests whether a file is writable.
static Path	move(Path source, Path target, CopyOption... options) Move or rename a file to a target file.
static BufferedReader	newBufferedReader(Path path, Charset cs) Opens a file for reading, returning a BufferedReader that may be used to read text from the file in an efficient manner.
static BufferedWriter	newBufferedWriter(Path path, Charset cs, OpenOption... options) Opens or creates a file for writing, returning a BufferedWriter that may be used to write text to the file in an efficient manner.
static SeekableByteChannel	newByteChannel(Path path, OpenOption... options) Opens or creates a file, returning a seekable byte channel to access the file.
static SeekableByteChannel	newByteChannel(Path path, Set<? extends OpenOption> options, FileAttribute<?>... attrs) Opens or creates a file, returning a seekable byte channel to access the file.
static DirectoryStream<Path>	newDirectoryStream(Path dir) Opens a directory, returning a DirectoryStream to iterate over all entries in the directory.
static DirectoryStream<Path>	newDirectoryStream(Path dir, DirectoryStream.Filter<? super Path> filter) Opens a directory, returning a DirectoryStream to iterate over the entries in the directory.
static DirectoryStream<Path>	newDirectoryStream(Path dir, String glob) Opens a directory, returning a DirectoryStream to iterate over the entries in the directory.
static InputStream	newInputStream(Path path, OpenOption... options) Opens a file, returning an input stream to read from the file.
static OutputStream	newOutputStream(Path path, OpenOption... options) Opens or creates a file, returning an output stream that may be used to write bytes to the file.
static boolean	notExists(Path path, LinkOption... options) Tests whether the file located by this path does not exist.
static String	probeContentType(Path path) Probes the content type of a file.
static byte[]	readAllBytes(Path path) Read all the bytes from a file.

static List<String>	readAllLines(Path path, Charset cs) Read all lines from a file.
static <A extends BasicFileAttributes> A	readAttributes(Path path, Class<A> type, LinkOption... options) Reads a file's attributes as a bulk operation.
static Map<String,Object>	readAttributes(Path path, String attributes, LinkOption... options) Reads a set of file attributes as a bulk operation.
static Path	readSymbolicLink(Path link) Reads the target of a symbolic link (optional operation).
static Path	setAttribute(Path path, String attribute, Object value, LinkOption... options) Sets the value of a file attribute.
static Path	setLastModifiedTime(Path path, FileTime time) Updates a file's last modified time attribute.
static Path	setOwner(Path path, UserPrincipal owner) Updates the file owner.
static Path	setPosixFilePermissions(Path path, Set<PosixFilePermission> perms) Sets a file's POSIX permissions.
static long	size(Path path) Returns the size of a file (in bytes).
static Path	walkFileTree(Path start, FileVisitor<? super Path> visitor) Walks a file tree.
static Path	walkFileTree(Path start, Set<FileVisitOption> options, int maxDepth, FileVisitor<? super Path> visitor) Walks a file tree.
static Path	write(Path path, byte[] bytes, OpenOption... options) Writes bytes to a file.
static Path	write(Path path, Iterable<? extends CharSequence> lines, Charset cs, OpenOption... options) Write lines of text to a file.

BaseFileAttributes

Modifier and Type	Method and Description
FileTime	creationTime() Returns the creation time.
Object	fileKey() Returns an object that uniquely identifies the given file, or null if a file key is not available.
boolean	isDirectory() Tells whether the file is a directory.
boolean	isOther() Tells whether the file is something other than a regular file, directory, or symbolic link.
boolean	isRegularFile() Tells whether the file is a regular file with opaque content.
boolean	isSymbolicLink() Tells whether the file is a symbolic link.
FileTime	lastAccessTime()

	Returns the time of last access.
FileTime	lastModifiedTime() Returns the time of last modification.
long	size() Returns the size of the file (in bytes).

FileSystems

Modifier and Type	Method and Description
static FileSystem	getDefault() Returns the default FileSystem.
static FileSystem	getFileSystem(URI uri) Returns a reference to an existing FileSystem.
static FileSystem	newFileSystem(Path path, ClassLoader loader) Constructs a new FileSystem to access the contents of a file as a file system.
static FileSystem	newFileSystem(URI uri, Map<String,?> env) Constructs a new file system that is identified by a URI
static FileSystem	newFileSystem(URI uri, Map<String,?> env, ClassLoader loader) Constructs a new file system that is identified by a URI

FileSystem

Modifier and Type	Method and Description
abstract void	close() Closes this file system.
abstract Iterable<FileStore>	getFileStores() Returns an object to iterate over the underlying file stores.
abstract Path	getPath(String first, String... more) Converts a path string, or a sequence of strings that when joined form a path string, to a Path.
abstract PathMatcher	getPathMatcher(String syntaxAndPattern) Returns a PathMatcher that performs match operations on the String representation of Path objects by interpreting a given pattern.
abstract Iterable<Path>	getRootDirectories() Returns an object to iterate over the paths of the root directories.
abstract String	getSeparator() Returns the name separator, represented as a string.
abstract UserPrincipalLookupService	getUserPrincipalLookupService() Returns the UserPrincipalLookupService for this file system (optional operation).
abstract boolean	isOpen() Tells whether or not this file system is open.
abstract boolean	isReadOnly() Tells whether or not this file system allows only read-only access to its file stores.
abstract WatchService	newWatchService() Constructs a new WatchService (optional operation).
abstract	provider()

FileSystemProvider	Returns the provider that created this file system.
abstract Set<String>	supportedFileAttributeViews() Returns the set of the names of the file attribute views supported by this FileSystem

FileStore

Modifier and Type	Method and Description
abstract Object	getAttribute(String attribute) Reads the value of a file store attribute.
abstract <V extends FileStoreAttributeView> V	getFileStoreAttributeView(Class<V> type) Returns a FileStoreAttributeView of the given type.
abstract long	getTotalSpace() Returns the size, in bytes, of the file store.
abstract long	getUnallocatedSpace() Returns the number of unallocated bytes in the file store.
abstract long	getUsableSpace() Returns the number of bytes available to this Java virtual machine on the file store.
abstract boolean	isReadOnly() Tells whether this file store is read-only.
abstract String	name() Returns the name of this file store.
abstract boolean	supportsFileAttributeView(Class<? extends FileAttributeView> type) Tells whether or not this file store supports the file attributes identified by the given file attribute view.
abstract boolean	supportsFileAttributeView(String name) Tells whether or not this file store supports the file attributes identified by the given file attribute view.

SimpleFileVisitor

Modifier and Type	Method and Description
FileVisitResult	postVisitDirectory(T dir, IOException exc) Invoked for a directory after entries in the directory, and all of their descendants, have been visited.
FileVisitResult	preVisitDirectory(T dir, BasicFileAttributes attrs) Invoked for a directory before entries in the directory are visited.
FileVisitResult	visitFile(T file, BasicFileAttributes attrs) Invoked for a file in a directory.
FileVisitResult	visitFileFailed(T file, IOException exc) Invoked for a file that could not be visited.

正規表示式	說明及範例	比對不成立之字串
/a/	含字母 “a” 的字串，例如 “ab”, “bac”, “cba”	“xyz”
/a./	含字母 “a” 以及其後任一個字元的字串，例如 “ab”, “bac” （若要比對.，請使用 \.）	“a”, “ba”
/^xy/	以 “xy” 開始的字串，例如 “xyz”, “xyab” （若要比對 ^，請使用 \^）	“axy”, “bxy”
/xy$/	以 “xy” 結尾的字串，例如 “axy”, “abxy” 以 “xy” 結尾的字串，例如 “axy”, “abxy” （若要比對 $，請使用 \$）	“xya”, “xyb”
[13579]	包含 “1〃 或 “3〃 或 “5〃 或 “7〃 或 “9〃 的字串，例如：” a3b”, “1xy”	“y2k”
[0-9]	含數字之字串	不含數字之字串
[a-z0-9]	含數字或小寫字母之字串	不含數字及小寫字母之字串
[a-zA-Z0-9]	含數字或字母之字串	不含數字及字母之字串
b[aeiou]t	“bat”, “bet”, “bit”, “bot”, “but”	“bxt”, “bzt”
[^0-9]	不含數字之字串（若要比對 ^，請使用 \^）	含數字之字串
[^aeiouAEIOU]	不含母音之字串（若要比對 ^，請使用 \^）	含母音之字串
[^\^]	不含 “^” 之字串，例如 “xyz”, “abc”	“xy^”, “a^bc”

.

正規表示式的特定字元	說明	等效的正規表示式
\d	數字	[0-9]
\D	非數字	[^0-9]
\w	數字、字母、底線	[a-zA-Z0-9_]
\W	非 \w	[^a-zA-Z0-9_]
\s	空白字元	[\r\t\n\f]
\S	非空白字元	[^ \r\t\n\f]

.

正規表示式	說明
/a?/	零或一個 a（若要比對? 字元，請使用 \?）
/a+/	一或多個 a（若要比對+ 字元，請使用 \+）
/a*/	零或多個 a（若要比對* 字元，請使用 *）
/a{4}/	四個 a
/a{5,10}/	五至十個 a
/a{5,}/	至少五個 a
/a{,3}/	至多三個 a
/a.{5}b/	a 和 b 中間夾五個（非換行）字元

.

字元	說明	簡單範例
\	避開特殊字元	/A*/ 可用於比對 “A*” ，其中 * 是一個特殊字元，為避開其特殊意義，所以必須加上 “\”
^	比對輸入列的啟始位置	/^A/ 可比對 “Abcd” 中的 “A” ，但不可比對 “aAb”
$	比對輸入列的結束位置	/A$/ 可比對 “bcdA” 中的 “A” ，但不可比對 “aAb”
*	比對前一個字元零次或更多次	/bo*/ 可比對 “Good boook” 中的 “booo” ，亦可比對 “Good bk” 中的 “b”
+	比對前一個字元一次或更多次，等效於 {1,}	/a+/ 可比對 “caaandy” 中的 “aaa” ，但不可比對 “cndy”
?	比對前一個字元零次或一次	/e?l/ 可比對 “angel” 中的 “el” ，也可以比對 “angle” 中的 “l”
.	比對任何一個字元（但換行符號不算）	/.n/ 可比對 “nay, an apple is on the tree” 中的 “an” 和 “on” ，但不可比對 “nay”
(x)	比對 x 並將符合的部分存入一個變數	/(a*) and (b*)/ 可比對 “aaa and bb” 中的 “aaa” 和 “bb” ，並將這兩個比對得到的字串設定至變數 RegExp.$1 和 RegExp.$2。
xy	比對 x 或 y	/a*b*/g 可比對 “aaa and bb” 中的 “aaa” 和 “bb”
{n}	比對前一個字元 n 次，n 為一個正整數	/a{3}/ 可比對 “lllaaalaa” 其中的 “aaa” ，但不可比對 “aa”
{n,}	比對前一個字元至少 n 次，n 為一個正整數	/a{3,}/ 可比對 “aa aaa aaaa” 其中的 “aaa” 及 “aaaa” ，但不可比對 “aa”
{n,m}	比對前一個字元至少 n 次，至多 m 次，m、n 均為正整數	/a{3,4}/ 可比對 “aa aaa aaaa aaaaa” 其中的 “aaa” 及 “aaaa” ，但不可比對 “aa” 及 “aaaaa”
[xyz]	比對中括弧內的任一個字元	/[ecm]/ 可比對 “welcome” 中的 “e” 或 “c” 或 “m”
[^xyz]	比對不在中括弧內出現的任一個字元	/[^ecm]/ 可比對 “welcome” 中的 “w”、” l”、” o” ，可見出其與 [xyz] 功能相反。（同時請注意 /^/ 與 [^] 之間功能的不同。）
[\b]	比對退位字元（Backspace character）	可以比對一個 backspace ，也請注意 [\b] 與 \b 之間的差別
\b	比對英文字的邊界，例如空格	例如 /\bn\w/ 可以比對 “noonday” 中的 ‘no’ ; /\wy\b/ 可比對 “possibly yesterday.” 中的 ‘ly’
\B	比對非「英文字的邊界」	例如, /\w\Bn/ 可以比對 “noonday” 中的 ‘on’ ， 另外 /y\B\w/ 可以比對 “possibly yesterday.” 中的 ‘ye’
\cX	比對控制字元（Control character），其中 X 是一個控制字元	/\cM/ 可以比對 一個字串中的 control-M
\d	比對任一個數字，等效於 [0-9]	/[\d]/ 可比對 由 “0〃 至 “9〃 的任一數字 但其餘如字母等就不可比對
\D	比對任一個非數字，等效於 [^0-9]	/[\D]/ 可比對 “w” “a” … 但不可比對如 “7〃 “1〃 等數字
\f	比對 form-feed	若是在文字中有發生 “換頁” 的行為 則可以比對成功
\n	比對換行符號	若是在文字中有發生 “換行” 的行為 則可以比對成功
\r	比對 carriage return	
\s	比對任一個空白字元（White space character），等效於 [\f\n\r\t\v]	/\s\w*/ 可比對 “A b” 中的 “b”
\S	比對任一個非空白字元，等效於 [^ \f\n\r\t\v]	/\S/\w* 可比對 “A b” 中的 “A”
\t	比對定位字元（Tab）	
\v	比對垂直定位字元（Vertical tab）	
\w	比對數字字母字元（Alphanumerical characters）或底線字母（” _” ），等效於 [A-Za-z0-9_]	/\w/ 可比對 “.A _!9〃 中的 “A”、” _” 、” 9〃。
\W	比對非「數字字母字元或底線字母」，等效於 [^A-Za-z0-9_]	/\W/ 可比對 “.A _!9〃 中的 “.” 、” ”、” !” ，可見其功能與 /\w/ 恰好相反。
\ooctal	比對八進位，其中 octal 是八進位數目	/\oocetal123/ 可比對 與 八進位的 ASCII 中 “123〃 所相對應的字元值。
\xhex	比對十六進位，其中 hex 是十六進位數目	/\xhex38/ 可比對 與 16 進位的 ASCII 中 “38〃 所相對應的字元。

我們知道要如何讀寫檔案文字

但是如果我們要讀寫的是數字、浮點數甚至是物件呢？

如果我們要讀寫的是數字、浮點數甚至是物件呢？

當然你可以把 123 或者 0.01 還有其他資料用 String 的方式寫到檔案中，就像剛才那樣

不過如此寫入的話，取出時，還要做字串分析等等繁雜的工作

雖然不是不可以，但是程式就不漂亮了

那麼，該如何把真正的"資料型態"寫到檔案裡面呢？

我們需要一些轉接頭，把資料型態與串流做個連結，稱作 chain

為了達到串流連結，我們要用到一些的類別，也就是轉接頭，稱作 Filter

Filter 除了可以達到轉換型態的功能外，還有些具有緩衝的功能，也就是讓流去的串流可以被回朔

來看看 JAVA 提供那些 Filter 的類別

緩衝區	**BufferedInputStream** **BufferedOutputStream** **BufferedReader** **BufferedWriter**
物件串流	**ObjectInputStream** **ObjectOutputStream**
基本資料型態串流	InputStreamReader InputStreamWriter **DataInputStream** **DataOutputStream**
計算行數	LineNumberInputStream LineNumberReader
堆疊串流	PushbackInputStream PushbackReader
列印串流	PrintStream PrintWriter

藍色的比較常用喔

不過常用的也不會很多，拿比較常用的 DataOutputStream 舉例吧

DataOutputStream 可以將輸出的資料轉換成各種基本資料型態：int, char, double, float...等等

這樣就完成了寫入 short 資料型態的數字到檔案裡了，當然也可以用類似方式寫入整數、浮點數等等...

不過當好奇的用記事本打開檔案一看...奇怪，啥都沒有啊！

那是當然的了，因為那不是以"字串"的編碼寫入，而是以你指定的資料編碼寫入(上面例子就是 short)

而記事本只會把資料以字串的編碼來呈現，其中字串編碼又分 ASCII、UTF...數種

不過當你再寫一段以 short 讀入資料的程式之後，你就可以直接用 short 的變數來接收讀入的資料了

超方便的！

//

http://tw.gitbook.net/java/io/fileinputstream_read.html

java.io.FileInputStream.read() 讀取當前輸入流中一個字節的數據

一般小的字节(byte)文件的读取和写入数据，我们使用 FileInputStream 和 FileOutputStream 类就可以实现了，但是对于大的字节(byte)文件的读取和写入数据，性能就会有很大的问题，我们一般选择 BufferedInputStream 和 BufferedOutputStream 来处理，也就是说

BufferedInputStream 和 BufferedOutputStream 主要的目的是提高字节(byte)文件的内容读取和写入的性能。

BufferedInputStream 和 BufferedOutputStream 是过滤流,需要使用已存在的节点来构造,即必须先有 InputStream 或 OutputStream,相对直接读写,这两个流提供带缓存的读写,提高了系统读写效率性能.BufferedInputStream 读取的是字节 byte,因为一个汉字占两个字节,而当中英文混合的时候,有的字符占一个字节,有的字符占两个字节,所以如果直接读字节,而数据比较长,没有一次读完的时候,很可能刚好读到一个汉字的前一个字节,这样,这个中文就成了乱码,后面的数据因为没有字节对齐,也都成了乱码.所以我们需要用 BufferedReader 来读取,它读到的是字符,所以不会读到半个字符的情况,不会出现乱码.