Java I/O Fundamentals

Chapter 23

I/O Streams

I/O Streams = sequence of data which is the content of a file.

When we **read** that sequence of bytes from a file,

we are reading an ***input stream***.

When we **write** that sequence of bytes to a file,

we are writing an ***output stream***.

Files

Files and directories are managed by a *file system*.

java.io.File class represents either a file or a directory path of a file system.

```
File file = new File("/home/user.properties");
```

You're not creating a new file, you are just creating an object

that may represent an actual file or directory (it may not even exist yet).

```java
public static void main(String[] args) {
    createPropertyFile();
    File file = new File(getCurrentPath() + "/home/user.properties");

    if (file.exists()) {                            // file or directory exist

        String name = file.getName();               // Name of the file/directory

        String parent = file.getParent();           // Path of its parent

        long millis = file.lastModified();          // Returning the time the file/directory was modified
                                                    // in milliseconds since 00:00:00 GMT, January 1, 1970

        if (file.isFile()) {                        // If the object represents a file
            long size = file.length();              // Returning the size of the file in bytes

        } else if (file.isDirectory()) {            // If the object represents a directory
            boolean dirCreated = file.mkdir();      // Returns true only if the directory was created

            boolean dirsCreated = file.mkdirs();    // Returns true only if the directory was created,
                                                    // along with all necessary parent directories

            String[] fileNames = file.list();       // Get all the files/directories in a directory, Just the names

            File[] files = file.listFiles();        // As File instances
        }
        boolean wasRenamed = file.renameTo(new File("user2"));  //return true if and only if the renaming succeeded;
        boolean wasDeleted = file.delete();         //return true if and only if the file or directory is successfully deleted
    }
}
```

java.io Package

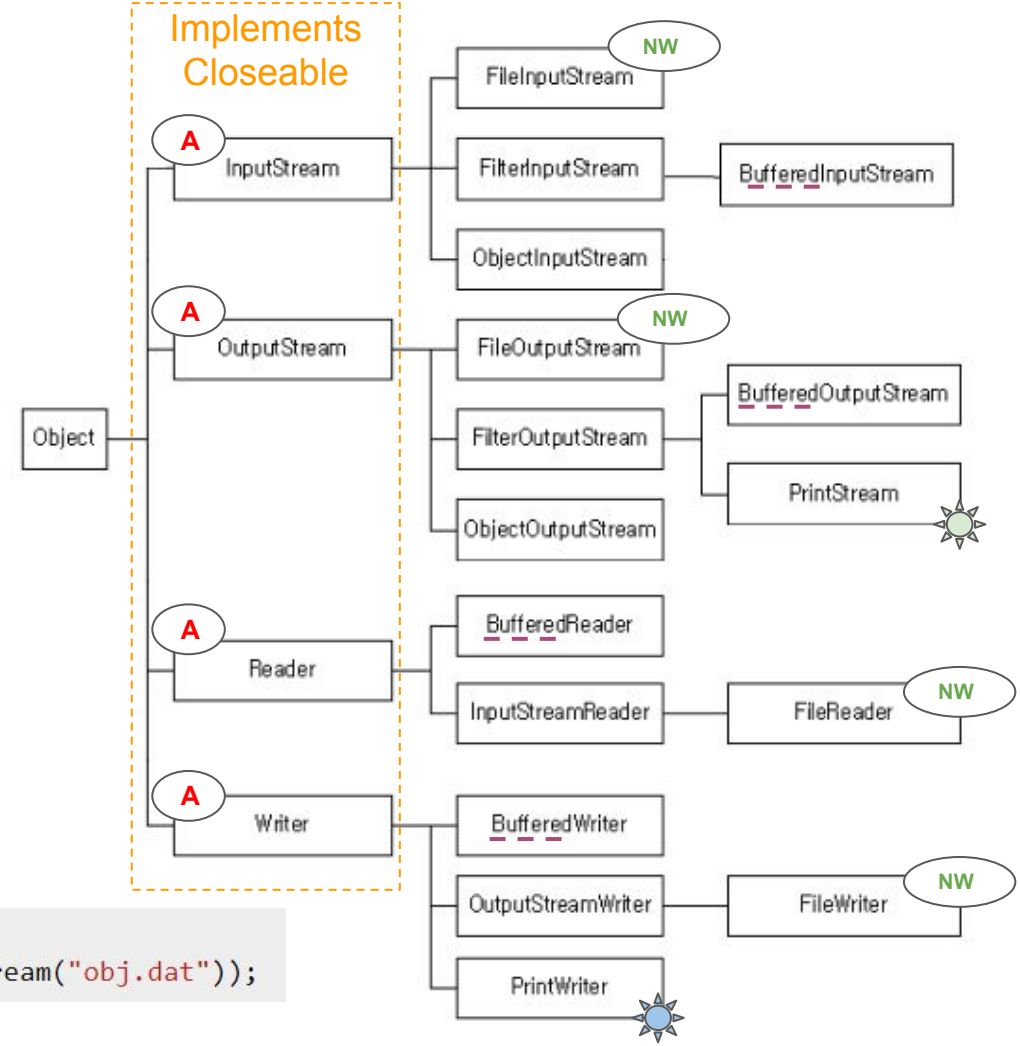
Stream in their name read or write streams of **BYTES**

Reader or **Writer** in their name read or write streams of **CHARACTERS**

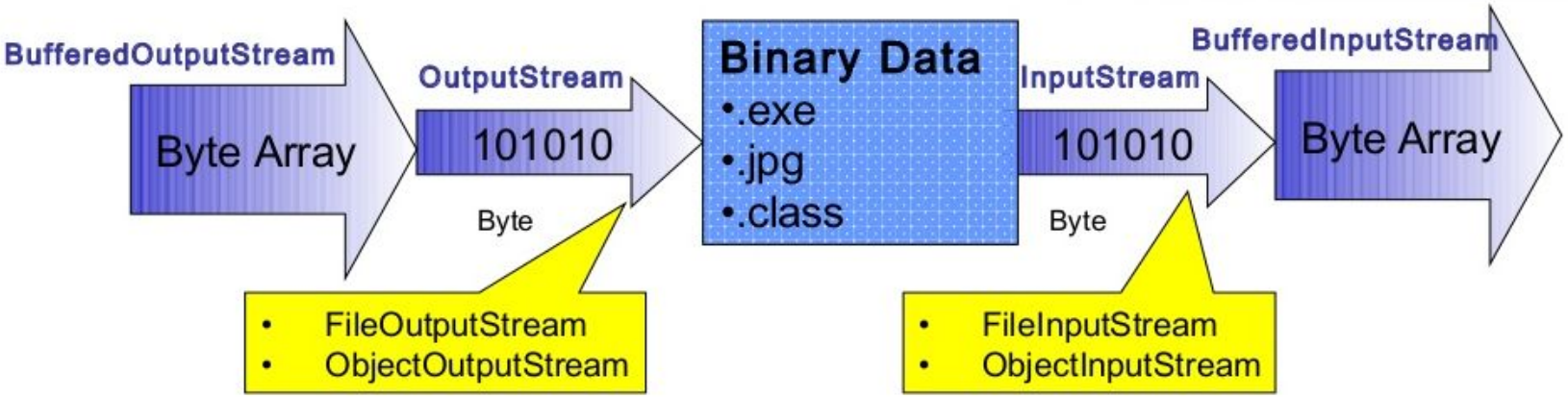
Buffered in their name, which use a buffer to **read or write data in groups** (of bytes or characters) to do it more efficiently.

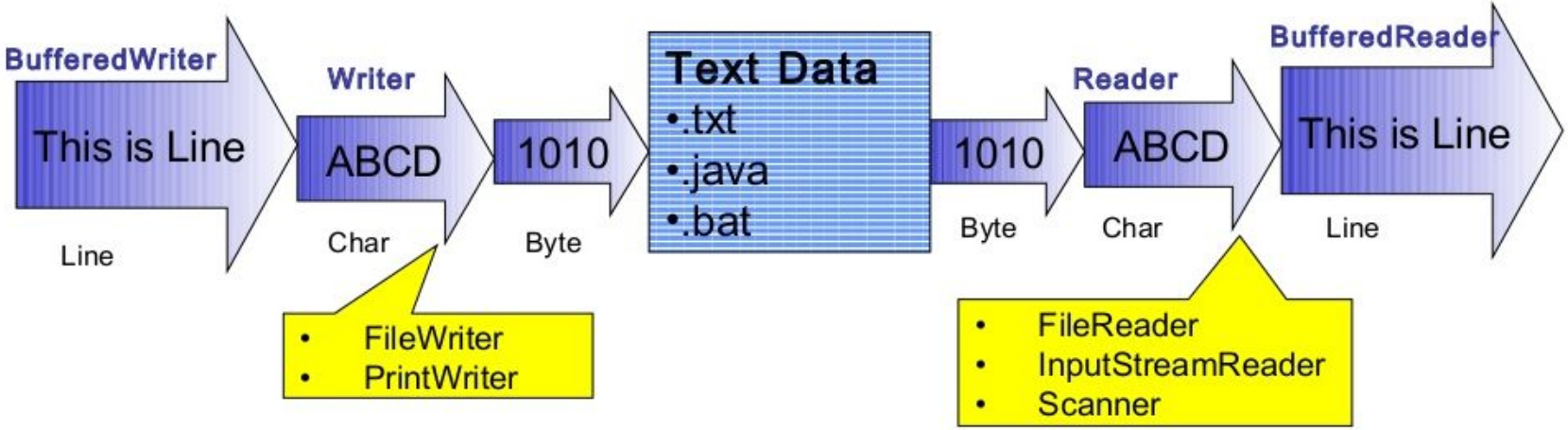
A Abstract Class

NW Non-Wrapper Class




```
ObjectInputStream ois =
    new ObjectInputStream(new FileInputStream("obj.dat"));
```





Ref: https://www.slideshare.net/sunilos/java-input-output-and-file-handling

FileInputStream

```
FileInputStream(File file)
FileInputStream(String path)
```

file.txt

```
1 line1
2 line2
3 line3
4
```

```java
public static void main(String[] args) {
    File file = new File(Files01.getCurrentPath() + "/home/file.txt");
    try (InputStream in = new FileInputStream(file)) {
        int b;
        // read() = Reads the next byte of data from the input stream.
        while ((b = in.read()) != -1) { // -1 indicates the end of the file
            System.out.print(b + " ");
        }
    } catch (IOException e) {
    }
    System.out.println();
    try (InputStream in = new FileInputStream(Files01.getCurrentPath() + "/home/file.txt")) {
        byte[] data = new byte[1024];
        int numberOfBytesRead;
        // read(byte[] b) = Reads some number of bytes from the input stream and stores them into the buffer array
        while ((numberOfBytesRead = in.read(data)) != -1) {
            System.out.println("numberOfBytesRead : " + numberOfBytesRead);
        }
    } catch (IOException e) {
    }
}
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console | Search | Terminal

```
<terminated> IO01 [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Sep 17, 2017, 4:21:29 PM)
108 105 110 101 49 13 10 108 105 110 101 50 13 10 108 105 110 101 51
numberOfBytesRead : 19
```

FileOutputStream

```
FileOutputStream(File file)
FileOutputStream(File file, boolean append)
FileOutputStream(String path)
FileOutputStream(String path, boolean append)
```

boolean **append :** you want to overwrite or append to the file if it exists
(it's overwritten by default)

```
static int count = -1;
public static void main(String[] args) {
    File file = new File(Files01.getCurrentPath() + "/home/output.txt");
    try (OutputStream out = new FileOutputStream(file)) {
        int b;
        while ((b = getData()) != -1) {
            out.write(b);    // Writes b to the file output stream
            out.flush();     // Flushes this output stream and forces any buffered output bytes to be written out.
        }
    } catch (IOException e) {
    }
}

private static int getData() {
    count++;
    if(count > 5)
        return -1;
    return 65 + count;
}
```

output.txt

```
1 ABCDEF
```

FileReader

```
FileReader(File file)
FileReader(String path)
```

```java
public static void main(String[] args) {
    File file = new File(Files01.getCurrentPath() + "/home/file.txt");
    try (Reader r = new FileReader(file)) {
        int c;
        //read() = Reads a single character.
        while((c = r.read()) != -1) { // -1 indicates the end of the file
            char character = (char)c;
            System.out.print(character);
        }
    } catch(IOException e) { /** ... */ }
    System.out.println("-----");
    try (Reader r = new FileReader(Files01.getCurrentPath() + "/home/file.txt")) {
        char[] data = new char[1024];
        int numberOfCharsRead;
        //read(char[] cbuf) = Reads characters into an array.
        while((numberOfCharsRead = r.read(data)) != -1) {
           System.out.println(data);
        }
    } catch (IOException e) {
    }
}
```

file.txt

```
1 line1
2 line2
3 line3
4
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

```
<terminated> IO03 [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Sep 17, 2017
line1
line2
line3
-----
line1
line2
line3
```

FileWriter

```
FileWriter(File file)
FileWriter(File file, boolean append)
FileWriter(String path)
FileWriter(String path, boolean append)
```

boolean **append :** you want to overwrite or append to the file if it exists (it's overwritten by default)

```
public static void main(String[] args) {
    File file = new File(Files01.getCurrentPath() + "/home/output.txt");
    try (Writer w = new FileWriter(file)) {
        w.write('-');                          // writing a character
        w.write("Writing to the file...");
        w.flush();
    } catch(IOException e) { /** ... */ }
}
```

output.txt

```
1 -Writing to the file...
```

BufferedReader

```
BufferedReader(Reader in)
BufferedReader(Reader in, int size)
```

Rather than read one character at a time, BufferedReader **reads a large block at a time** into a buffer.

The buffer size here refers to the number of bytes it buffers.
Default Value = 8192 bytes

```java
public static void main(String[] args) {
    File file = new File(Files01.getCurrentPath() + "/home/file.txt");
    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        String line;
        while ((line = br.readLine()) != null) {    // null indicates the end of the file
            System.out.println(line);
        }
    } catch (IOException e) {
    }
}
```

file.txt

```
1 line1
2 line2
3 line3
4
```

Console | Markers | Properties | Servers

```
<terminated> Wrapper01 [Java Application] C:\Program Files\J
line1
line2
line3
```

When the BufferedReader is closed,
it will also close the Reader instance it reads from.

BufferedWriter

```
BufferedWriter(Writer out)
BufferedWriter(Writer out, int size)
```

The buffer size here refers to the number of bytes it buffers.
Default Value = 8192 bytes

```java
public static void main(String[] args) {
    File file = new File(Files01.getCurrentPath() + "/home/output.txt");
    try (BufferedWriter bw = new BufferedWriter(new FileWriter(file))) {
        bw.write("Writing to the file...");
        bw.flush();
    } catch (IOException e) {

    }
}
```

output.txt

```
1 Writing to the file...
```

When the BufferedWriter is closed, it will also close the Writer instance it writes to.

ObjectInputStream / ObjectOutputStream

The process of **converting an object to a data format** that can be stored (in a file for example) is called ***serialization*** and **converting that stored data format into an object** is called ***deserialization***.

If you want to serialize an object, its class must **implement the java.io.Serializable** interface
If you try to serialize a class that doesn't implement that interface,
a **java.io.NotSerializableException** will be thrown at runtime.

ObjectOutputStream allows you to **serialize** objects to an OutputStream
ObjectInputStream allows you to **deserialize** objects from an InputStream.

When serialization ?

- When an object persisted in a file.
- When an object sent over the Network.
- When an object sent to Hardware.
- Or in the other words when an object is sent Out of JVM.

ObjectOutputStream

```
ObjectOutputStream(OutputStream out)
```

```java
//ObjectOutputStream
public class Wrapper03 {
    public static void main(String[] args) {
        File file = new File(Files01.getCurrentPath() + "/home/obj.dat");
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(file))) {
            Box box = new Box();
            oos.writeObject(box);
        } catch (IOException e) {
            /** ... */
        }
    }
}

class Box implements java.io.Serializable {
    public String name = "Box";
}
```

obj.dat

```
1 ¬í ▯sr <com.wealth.certificate.study_1z0_809.chapter23.iowrapper.BoxN▯
2 ♦M♦♥O▯ ▯L ▯namet ▯Ljava/lang/String;xpt ▯Box
```

ObjectInputStream

```
ObjectInputStream(InputStream in)
```

```java
//ObjectInputStream
public class Wrapper04 {
    public static void main(String[] args) {
        File file = new File(Files01.getCurrentPath() + "/home/obj.dat");
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {
            Box box = null;
            Object obj = ois.readObject();
            if (obj instanceof Box) {
                box = (Box) obj;
                System.out.println(box.name);
            }
        } catch (IOException ioe) {
        } catch (ClassNotFoundException cnfe) {
        }
    }
}
```

```
Console   Markers   Properties   Serv
<terminated> Wrapper04 [Java Application] C:\Prograr
Box
```

Two important notes. When deserializing an object,

1. the constructor, and any initialization block are not executed,
2. null objects are not serialized/deserialized.

PrintWriter

PrintWriter is a subclass of Writer that writes formatted data to another (wrapped) stream, even an OutputStream.

```
PrintWriter(File file)
    throws FileNotFoundException
PrintWriter(File file, String charset)
    throws FileNotFoundException, UnsupportedEncodingException
PrintWriter(OutputStream out)
PrintWriter(OutputStream out, boolean autoFlush)
PrintWriter(String fileName) throws FileNotFoundException
PrintWriter(String fileName, String charset)
    throws FileNotFoundException, UnsupportedEncodingException
PrintWriter(Writer out)
PrintWriter(Writer out, boolean autoFlush)
```

By default, it uses the default charset of the machine but at least, this class accepts the following charsets (there are other optional charsets):

- US-ASCII
- ISO-8859-1
- UTF-8
- UTF-16BE
- UTF-16LE
- UTF-16

PrintWriter

```java
File file = new File(Files01.getCurrentPath() + "/home/printwriter.txt");
// Opens or creates the file without automatic line flushing
// and converting characters by using the default character encoding
try (PrintWriter pw = new PrintWriter(file)) {
    pw.write("Hi"); // Writing a String
    pw.write(100); // Writing a character

    // write the string representation of the argument
    // it has versions for all primitives, char[], String, and Object
    pw.print(true);
    pw.print(10);

    // same as print() but it also writes a line break as defined by
    // System.getProperty("line.separator") after the value
    pw.println(); // Just writes a new line
    pw.println("A new line...");

    // format() and printf() are the same methods
    // They write a formatted string using a format string,
    // its arguments and an optional Locale
    pw.format("%s %d", "Formatted string ", 1);
    pw.printf("%s %d", "Formatted string ", 2);
    pw.format(Locale.GERMAN, "%.2f", 3.1416);
    pw.printf(Locale.GERMAN, "%.3f", 3.1416);
} catch (FileNotFoundException e) {
    // if the file cannot be opened or created
}
```

printwriter.txt

```
1 Hidtrue10
2 A new line...
3 Formatted string  1Formatted string  23,143,142
```

Standard streams

Java initializes and provides three stream objects as `public static` fields of the `java.lang.System` class:

- `InputStream System.in`
  The standard input stream (typically the input from the keyboard)
- `PrintStream System.out`
  The standard output stream (typically the default display output)
- `PrintStream System.err`
  The standard error output stream (typically the default display output)

```
System.out.print("Enter a character: ");
try {
    int c = System.in.read();
} catch(IOException e) {
    System.err.println("Error: " + e);
}
```

```
BufferedReader br =
    new BufferedReader(new InputStreamReader(System.in));
String line = br.readLine();
// Or using the java.util.Scanner class
Scanner scanner = new Scanner(System.in);
String line = scanner.nextLine();
```

java.io.Console

Since Java 6, we have the java.io.Console class to access the console of the machine your program is running on.

But keep in mind that if you program is running in an environment that doesn't have access to a console (like an IDE or if your program is running as a background process), System.console() will return null.

```
Console console = System.console();
// Check if the console is available
if(console != null) {
    console.writer().println("Enter your user and password");
    String user = console.readLine("Enter user: ");
    // readPassword() hides what the user is typing
    char[] pass = console.readPassword("Password: ");
    // Clear password from memory by overwriting it
    Arrays.fill(pass, 'x');
}
```