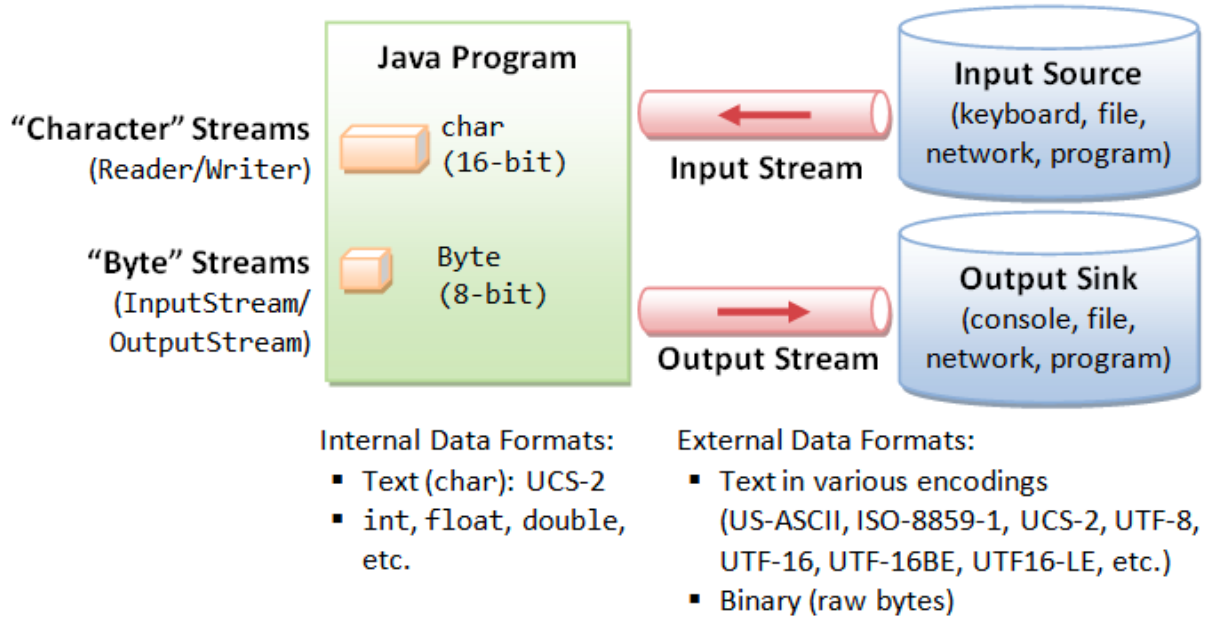
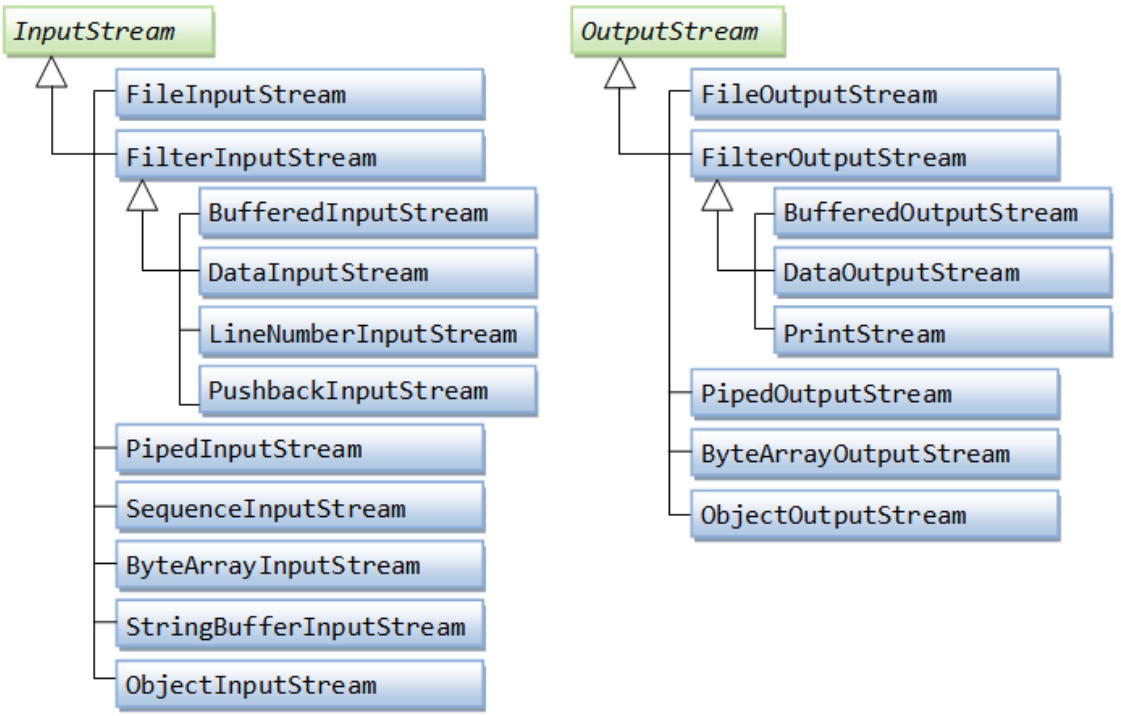
Input/Output

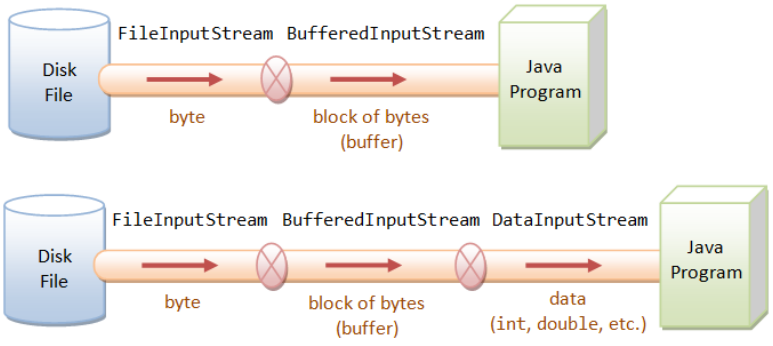


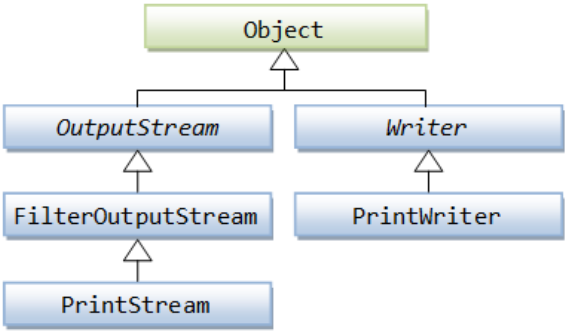
Internal Data Formats:
- Text (char): UCS-2
- int, float, double, etc.

External Data Formats:
- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

Flujo de Bytes - Imágenes, sonidos, formatos binarios en general







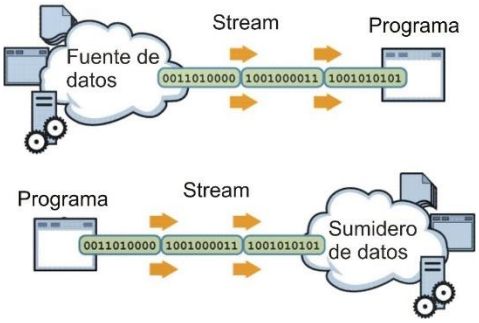
Filtros

Modo carácter - textos

1. Flujo de datos

Los flujos, *streams*, representan a un canal de comunicación por el que se envía una secuencia de datos. Los flujos poden ser de dos tipos en función de su origen:

- *Flujos de entrada*: proporciona información de entrada a nuestro programa.
- *Flujo de salida*: la secuencia de datos tiene como origen nuestro programa y como destino, un fichero u otra aplicación.



1.1. Tipos de flujos y clases

En Java existen dos tipos de flujos:

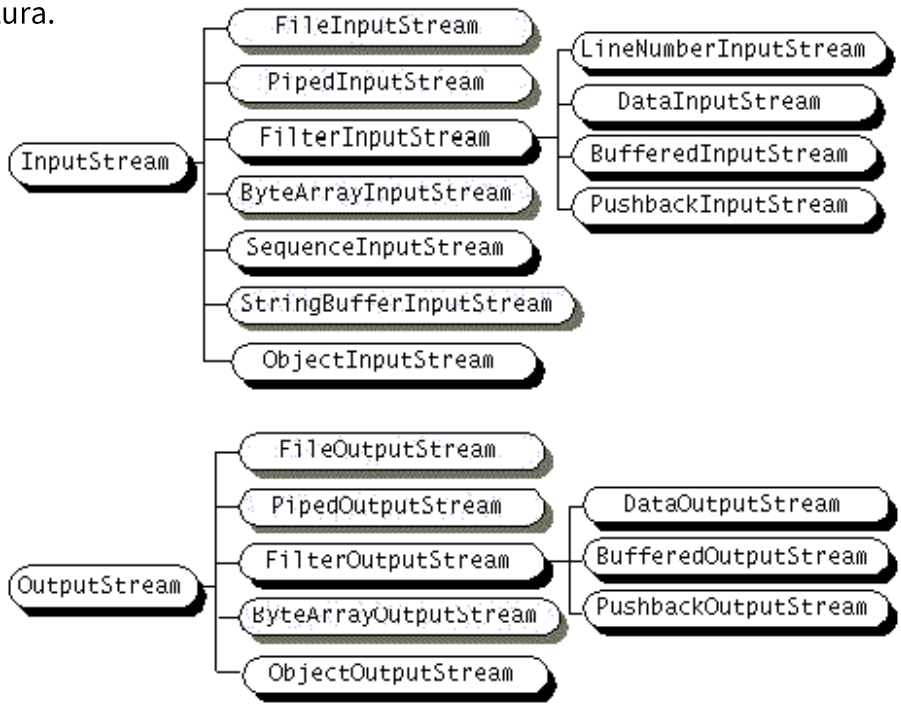
Flujos de bytes

Lo que se transmite a través del flujo son bytes, con lo que se va a poder recibir o enviar cualquier tipo de dato.

Es una forma de procesar la entrada y salida a un nivel más bajo, debe evitarse y usarlo para casos específicos.

Todas las clases relacionadas los flujos de bytes heredan de dos clases abstractas:

- *InputStream*: heredan las clases relacionadas con flujos de bytes que son de lectura.
- *OutputStream*: heredan las clases relacionadas con flujos de bytes que son de escritura.



Las clases orientadas a flujo de bytes

- ByteArrayInputStream y ByteArrayOutputStream

  Implementan un flujo de entrada y salida de bytes respectivamente, teniendo como fuente un array de bytes en el caso de la clase de entrada. En el caso de la clase que maneja la salida existen dos tipos de constructores: el 1º crea un buffer de 32 bytes y el 2º crea un buffer del tamaño que se le diga en el parámetro.

- FileInputStream

  Esta clase crea un *InputStream* que usaremos para leer los bytes del fichero. Define varios constructores de los cuales destacan el que le pasamos como parámetro un String que contiene la ruta completa del fichero y otro al que le pasamos un objeto de tipo File. En todos los casos, si el fichero no existe se debe lanzar la excepción *FileNotFoundException*. La sintaxis de los constructores más habituales es:

```
FileInputStream(String ruta_fichero);
FileInputStream(File fichero);                    // fichero es la descripción del archivo.
```

- FileOutputStream

  Este, al contrario que el anterior crea un *OutputStream* para enviar un flujo de bytes a un fichero. Igualmente, al constructor le podemos pasar la ruta completa del archivo o un objeto de tipo File. Dado que esta operación es de riesgo, la clase puede lanzar una *SecurityException*, también se pueden lanzar las excepciones *IOException* y/o *FileNotFoundException* si, por ejemplo, el fichero no se puede crear.

```
FileOutputStream(String ruta_fichero);
FileOutputStream(File fichero);                   // fichero es la descripción del archivo.
```

Flujos de caracteres

Lo que se transmite a través del flujo son caracteres. Es el tipo de flujo adecuado cuando queremos leer texto almacenado en un archivo.

Las clases relacionadas heredan de:

- *Reader*: para las clases relacionadas con flujos de caracteres que son de lectura.
- *Writer*: heredan las clases relacionadas con flujos de caracteres que son de escritura.

Las clases orientadas a flujo de caracteres

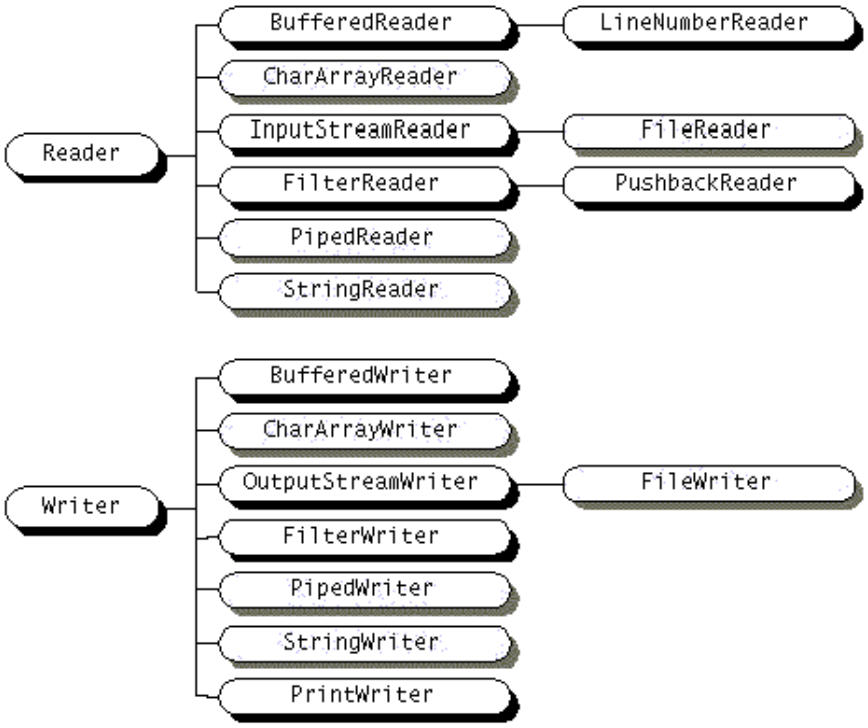
Como ya hemos visto, estas clases nacen de las clases abstractas Reader y Writer. Las clases concretas derivadas de ellas tienen su homónimo en las clases concretas derivadas de

InputStream y OutputStream, por tanto, la explicación hecha para todas ellas tiene validez para las orientadas a carácter salvo quizás algunos matices que podrás completar con las especificaciones por lo que omitiremos la explicación detallada de cada una de ellas. Las clases más importantes:

Acceso a fichero: FileReader y FileWriter.

Acceso a carácter: *CharArrayReader* y CharArrayWriter.

Buferización de datos: BufferedReader y BufferedWriter.



1.2. Uso de los flujos

Para usar los flujos hay un procedimiento genérico a seguir:

- *Abrir el flujo*, es el paso que establece la conexión con el dispositivo del que se va a leer o donde se van enviar los datos.
- *Caracterizar el flujo*, se le añaden las características al flujo que establecerá la forma de trabajar con el.
- *Leer o escribir datos* mientras haya información para leer o enviar.
- *Cerrar el flujo.* Los flujos estándar no son necesario cerrarlos, pero si creamos un flujo para leer de un fichero es necesario cerrar ese flujo.
- Un fallo en cualquier punto produce la excepción IOException.

La primera línea de este ejemplo establece un flujo de lectura desde el archivo "ejemplo.txt". La segunda, establece que para la lectura se va a utilizar un buffer, con lo que se harán un menor número de operaciones de entrada y salida en el fichero.

1.3. Aplicación de formatos de salida

Las clases PrintStream y PrintWriter implementan un conjunto de métodos para escribir tanto bytes como caracteres. Adicionalmente, estas clases permiten el formateo de los objetos.

Hemos utilizado los métodos print() y println() mediante una llamada desde el flujo estándar *System.out*, que es un objeto de la clase PrintStream, y que no permiten aplicar formatos a la salida.

Estas nuevas clases proporcionan un método adicional para poder aplicar un formato a la información de salida. La sintaxis del método es la siguiente:

```
String s = "Hola";
float numero = 43.567756456463f;
System.out.format("Sin formato: %2$f , con formato: %2$+05.2f %n", s,
numero);
```

PrintWriter format(*String formato*, *Object*... *argumentos*)

Donde:

- *formato*: es una cadena de caracteres que tiene una sintaxis muy concreta donde se define el formato que van a tener los datos que se impriman.
- *argumentos*: son una lista de argumentos separados por comas a los que se hace referencia en el parámetro formato. En caso de que haya más argumentos de los referenciados en formato, estos serán ignorados.

Lo más importante de este método es generar la cadena de texto en la que se indica el formato de salida, es una cadena de texto normal que lleva incrustados una serie de marcadores de formato.

Formato modificadores

- Para caracteres y números

  %[indiceArgumento$][flags][ancho][.precisión]conversión
- Para fechas y horas

%[indiceArgumento$][flags][ancho]conversión

- ✓ *indiceArgumento*: es opcional e indica la posición del argumento a formatear dentro de la lista de argumentos. Para hacer referencia al primer argumento de la lista se usa "1$", para el segundo "2$", y así sucesivamente.
- ✓ *flags*: también es opcional. Consiste en un conjunto de caracteres que modifican el formato de salida. Este conjunto depende do que haya en conversión.
- ✓ *ancho*: también opcional. Es un número entero positivo que indica el número mínimo de caracteres que serán escritos en la salida.
- ✓ *precisión*: opcional. Es un entero no negativo que normalmente se usa para restringir el número de caracteres en la salida. Su comportamiento se adapta al que haya en conversión.
- ✓ *conversión*: es obligatorio y se trata de un carácter como debería ser formateado el argumento. En el caso de las fechas y horas, son dos caracteres: el primero será una 't' o 'T' y el segundo carácter indicará el formato que será usado.

Flags

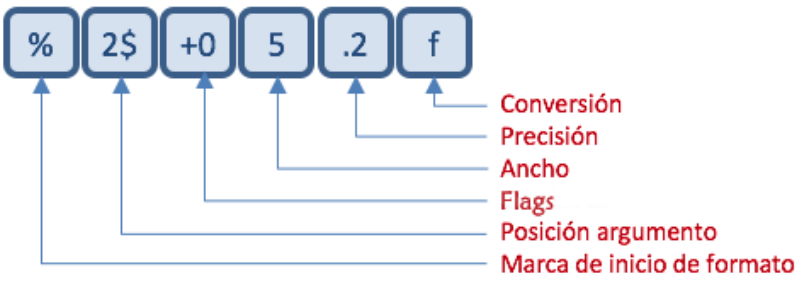
Caracter	*Significado*
-	El resultado saldrá justificado.
#	Formatea en una forma alternativa para el marcador de formato.
+	El resultado siempre incluirá el signo.
espacio	El resultado incluirá un espacio para valores positivos.
0	El resultado se cubrirá con 0 hasta la longitud especificada.
,	Añade un separador de grupos. Aplicado solo a números.
(	El resultado cerrará entre paréntesis números negativos.
<	Para reusar el valor del argumento utilizado en el anterior marcador de formato. Usado sobre todo en el formateo de fechas y horas

Conversiones

Caracter	*Descripción*
B / b	Para representar a un booleano.
S / s	Para representar a un string.
C / c	Para representar a un carácter.
d	Para representar a un entero.
f	Para representar a un número decimal.
%	Para representar al literal “%”
n	Para representar a un separador de línea.

Segundo carácter para fechas y horas

Carácter	*Descripción*
H	Para representar la hora en formato de 24 horas y con dos dígitos.
I	Para representar la hora en formato de 12 horas y con dos dígitos.
M	Para representar los minutos con dos dígitos.
S	Para representar os segundos con dos dígitos.
y	Para representar el ano con dos dígitos.
m	Para representar el mes, formateado con dos dígitos.
h	Para representar el mes en texto abreviado.
B	Para representar el mes en letras.
d	Para representar el día, formateado con dos dígitos.



Referenciando al segundo argumento (2$), que se va a mostrar con signo (+). Muestra 5 dígitos (5) de la parte entera, si hay menos se rellenan con 0 (0), solo 2 dígitos de la parte decimal, es la conversión de un número flotante (f).

Otro método que proporcionan estas clases y que permite el formateado de salida es el método *printf()*. Su funcionamiento es exactamente igual a format().