Entrada salida standard

Entrada estándar: teclado.
Salida Standard: pantalla

Los dispositivos estándar se pueden cambiar, por ejemplo podemos derivar la salida a un fichero, pero no nos pararemos con esos detalle en este boletín.

El objeto que maneja el teclado lo crea automáticamente la MVJ al iniciar la aplicación. Hay un objeto por aplicación, no tenemos que hacer un new de ninguna clase para acceder al teclado. Concretamente manejamos el teclado a través de un objeto referenciado por la variable referencia "System.in". Si consultamos el API JAVA vemos que

in

```
public static final InputStream in
```

The "standard" input stream. This stream is already open and ready to supply input data. Typically this stream corresponds to keyboard input or another input source specified by the host environment or user.

Observa que:

- InputStream es abstracta. Si por curiosidad quieres saber qué clase concreta está usando System.in

```
class App {
    public static void main(String[] args) {
        System.out.println(System.in.getClass());
    }
}
```

- la variable *in* es *final*

No puedo hacer System.in=otro objeto. La inicializa el sistema por primera vez y por ser final no se le puede asignar luego otro objeto.

- InputStream es un flujo de bytes. No existe en el API java una clase que me permita acceder al teclado como un flujo de caracteres de forma directa. Tendré que hacer conversiones de flujo explícitas como veremos en los ejemplos.

EL BUFFER DE TECLADO

leer un byte con read()

Es la forma más básica de leer del teclado
Ejemplo: Leer caracteres por teclado hasta teclear 'q'

```
class App{
   public static void main(String[] args) throws java.io.IOException{
      char c;
      do{
         System.out.println("Oprime una tecla y luego ENTER");
         c=(char)System.in.read();
      }while(c!='q');
   }
}
```

Si ejecutamos el programa anterior, la salida por pantalla puede resultar confusa. Por ejemplo,

```
run:
Oprime una tecla y luego ENTER
hola
Oprime una tecla y luego ENTER
Oprime una tecla y luego ENTER
Oprime una tecla y luego ENTER
Oprime una tecla y luego ENTER
Oprime una tecla y luego ENTER
h
Oprime una tecla y luego ENTER
Oprime una tecla y luego ENTER
q
BUILD SUCCESSFUL (total time: 29 seconds)
```

El funcionamiento del programa anterior está condicionado por:

1. El código anterior está escrito para que el usuario vaya realmente introduciendo de carácter en carácter (1 caracter + enter, 1 caracter + enter, …).
2. La entrada estándar (teclado) funciona con buffer, esto es, con una memoria intermedia donde se almacenan los caracteres tecleados.

Utilizar buffer tiene ventajas, por ejemplo, si tecleamos algo mal, podemos borrar y rectificar, ya que, el método read() no comienza a 'leer' los caracteres del buffer hasta que el usuario teclea ENTER. ENTER es la señal que le indica a System.in que puede empezar a pasar los caracteres del buffer hacia el programa.
Pulsar la tecla ENTER genera un carácter salto de línea '\n' que también se almacenan en el buffer como otro carácter cualquiera. Según versión de S.O. puede ocurrir que la tecla enter genera dos caracteres, \n y \r.
Si tecleamos *HOLA* y luego pulsamos ENTER en el buffer de entrada tendremos almacenado:

H	O	L	A	\n

Y si tecleamos simplemente H y luego ENTER

H	\n

Por lo tanto, si introducimos HOLA<ENTER> con el programa anterior vemos la frase
"Oprime una tecla y luego ENTER"
5 veces.
Y si tecleamos H<ENTER> vemos la frase 2 veces.

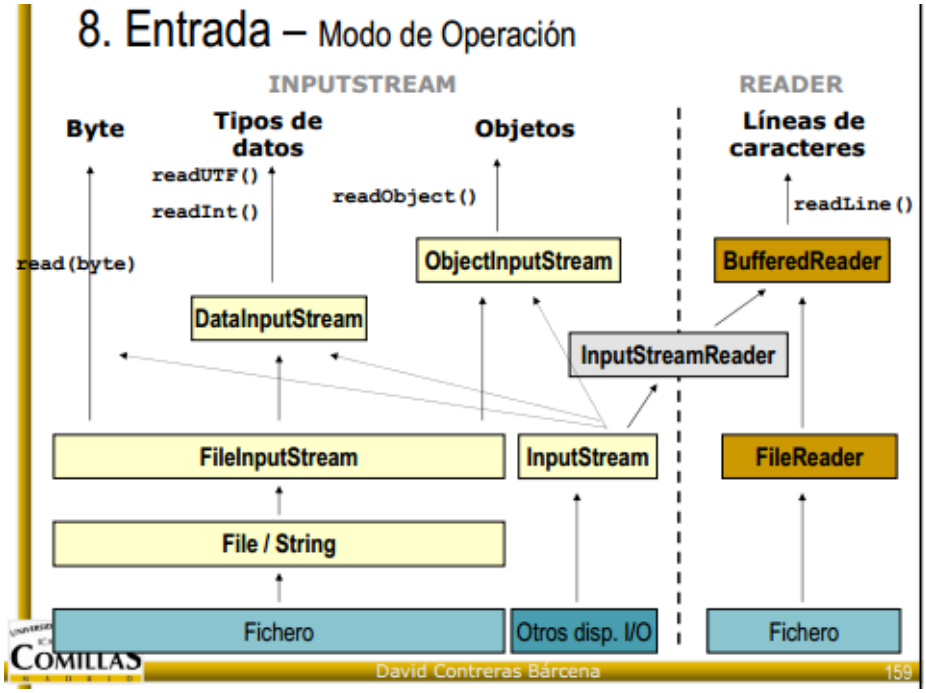
Leer del teclado con objetos *flujos de caracteres*

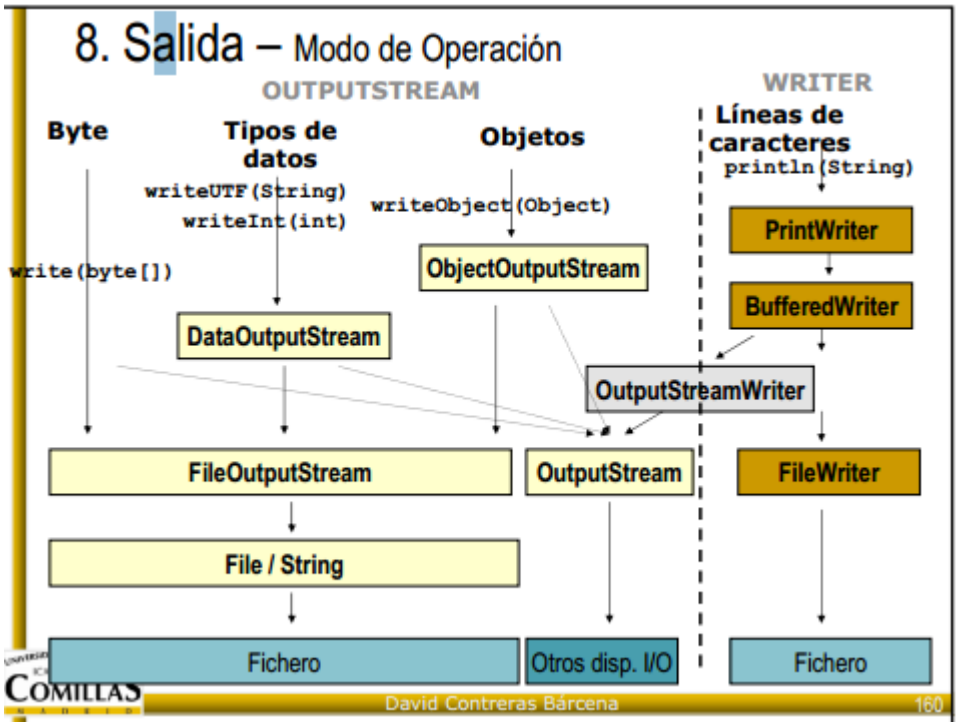
System.in es un flujo de bytes¿Por que me empeño en utilizar un flujo de caracteres?. Porque las clases basadas en flujos de caracteres disponen de un método readLine() que permite leer líneas

De http://www.labcom.upcomillas.es/poo/itig/apuntes/Java08.pdf

Sacamos estos dos interesantes gráficos.

En el dibujo fíjate bien en el método readLine() ¡es el método que queremos usar!, pero … es un método que pertenece a los objetos que son flujos de caracteres. No obstante, es posible si nos interesa, "convertir" un flujo de bytes en flujo de caracteres a través de clases intermediarias como InputStreamReader.

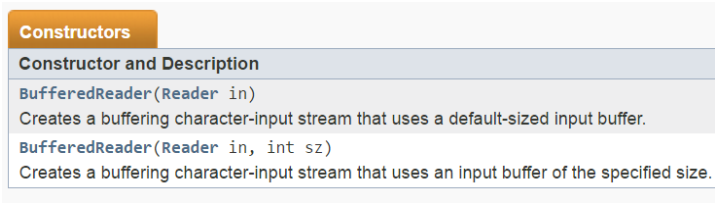




Conclusión: para leer del teclado tengo que usar un InputStream, es decir, un flujo de bytes, pero si me interesa, ese flujo de bytes se puede convertir en flujo de caracteres a través de la clase puente InputStreamReader. Para la escritura hay una conclusión simétrica.

LA CLASE BufferedReader y EL MÉTODO readLine() para leer líneas enteras

Leer de carácter en carácter se adecúa bien sólo a ciertos contextos, normalmente es más práctico para los programas que procesan texto leer de línea en línea con el método readLine() de la clase BufferedReader



Constructors

Constructor and Description
`BufferedReader(Reader in)` Creates a buffering character-input stream that uses a default-sized input buffer.
`BufferedReader(Reader in, int sz)` Creates a buffering character-input stream that uses an input buffer of the specified size.

Para leer del teclado, la clase que se usaba, antes de ser eclipsada por la clase Scanner era *BufferReader. E*sta clase no se puede conectar directamente con el objeto System.in ya que System.in es un flujo de bytes y el constructor BufferReader precisa un flujo Reader(de caracteres). Es necesario utilizar una clase "utilidad" intermedia que convierta el flujo de bytes System.in en flujo de caracteres: la clase InputStreamReader.

El código que hace lo anterior podría ser:
BufferedReader br = new BufferedReader(new InputStreamReader(System.in))

Ejemplo: Ejecuta el siguiente código de lectura de cadenas

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
class App{
   public static void main(String[] args) throws IOException{
      BufferedReader br= new BufferedReader( new InputStreamReader(System.in));
      String cad;
      System.out.println("teclea frases, frase \"alto\" para fin");
      do{
        cad=br.readLine();
        System.out.println(cad);
      }while(!cad.equals("alto"));
   }
}
```

La firma de readLine() es:

*public String **readLine**() throws IOException*

Readline() lee líneas de texto del buffer. Una "línea de texto" se refiere a un conjunto de caracteres que terminan en /n o en /r o en /n/r. Una vez que lee la línea del buffer, devuelve un string con todos los caracteres pero sin los terminadores.

Si la instrucción
BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
Te resulta muy compacta puedes desglosarla en dos:
InputStreamReader isr= new InputStreamReader(System.in);//pasa flujo de byte a caracteres
BufferedReader br= new BufferedReader(isr);//flujo de caracteres con buffer y que tiene método readline

OJO CON CERRAR EL FLUJO DEL TECLADO
Es una buena práctica cerrar todo flujo que se abre. Imagina un código que abre dentro de un bucle un recurso y no lo cierra. Imagina a continuación que el bucle es un while true, acabará colapsando la RAM asignada al programa. Los recursos asociados a ficheros y bases de datos también suelen trabajar con buffers que actualizan el disco al ser cerrados.

El caso del teclado es distinto ya que System.in es final y siempre habrá un único objeto

teclado por aplicación y si lo cerramos en mitad de la aplicación perdemos de forma irremediable la conexión con el teclado ya que System.in es final y no se puede volver a instanciar.

Ejemplo: comprueba que el último readline() genera una excepción

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
class App{
   public static void main(String[] args) throws IOException{
      InputStreamReader isr= new InputStreamReader(System.in);
      BufferedReader br= new BufferedReader(isr);
      String cad;
      System.out.println("teclea una frase");
      cad=br.readLine();
      System.out.println("tecleaste: "+cad);
      System.in.close();
      System.out.println("teclea otra frase");
      cad=br.readLine();
      System.out.println("tecleaste: "+cad);
   }
}
```

Sobre Scanner(System.in)

Cuando usamos Scanner(System.in) OCURRE LO MISMO, no cierres el Scanner asociado al teclado hasta que sepas con certeza no vas a volver a requerir el teclado hasta el fin de la aplicación, o simplemente, no lo cierres nunca a no ser que se trate de una aplicación muy grande y compleja con hilos y mil complicaciones que son casos especiales.

Ejercicio U9_B2_E1: Escribe un ejemplo que demuestre que si cierro un Scanner asociado a System.in, también cierro System.in y no puedo volver a usar el teclado en la aplicación

LEER VALORES NUMÉRICOS DE TECLADO CON readLine()

Ya sabemos hacerlo con los métodos de Scanner. Con readline() leemos un String y luego convertimos el String a número utilizando clases envolventes de tipos primitivos

Recuerda que cada tipo primitivo tiene su correspondiente clase envoltorio.

- Byte para byte.
- Short para short.
- Integer para int.
- Long para long.
- Boolean para boolean
- Float para float.
- Double para double y
- Character para char.

Todas las clases envoltorio tienen un método parseTipo() para convertir un String en número.

Ejemplo:

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
class App{
   public static void main(String[] args) throws IOException{
      BufferedReader br=new BufferedReader( new InputStreamReader(System.in));
      String str;
      double promedio=0.0;
      int n; //cantidad de números que vamos a introducir
      System.out.println("¿Cuántos números quieres introducir?");
```

```
        str=br.readLine();
        n=Integer.parseInt(str);
        for(int i=0;i<n;i++){
          System.out.print("numero " + i +": ");
          str=br.readLine();
          promedio+=Double.parseDouble(str);
          System.out.println();
        }
        promedio/=n;
        System.out.println("El promedio es: " + promedio);
        br.close();
    }
}
```

BufferedReader vs Scanner

En los primeros JDK no había clase Scanner y la lectura de teclado sólo era posible con BufferedReader. En JDK subsiguientes apareció la clase Scanner y entre sus muchos usos se extendió rápidamente su uso para leer de teclado por que es más cómoda y no es necesario tener en mente el concepto de flujo de bytes/caracteres.

No obstante BufferedReader al ser de más bajo nivel en ciertos contextos es más flexible, por ejemplo al trabajar con hilos. A continuación vemos un ejemplo de cómo con BufferedReader a través del método ready() podemos conseguir que la espera de que el usuario introduzca algo por teclado y lo envíe con enter esté limitado a cierta cantidad de tiempo.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.concurrent.TimeoutException;
public class App{
  public static String readLineTimeout(BufferedReader reader, long timeout) throws TimeoutException, IOException {
      long start = System.currentTimeMillis();
       while (!reader.ready()) {
          if (System.currentTimeMillis() - start >= timeout)
              throw new TimeoutException();
           // optional delay between polling
          try { Thread.sleep(50); } catch (Exception ignore) {}
      }
       return reader.readLine(); // won't block since reader is ready
  }
  public static void main( String[] args ){
      BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
      String leido="";
      String fueraDeTiempo="\nlo siento, pasaron mas de 10 segundos ";
      boolean antesDe10=true;
      try {
          System.out.print("teclea algo y pulsa enter antes de 10 segundos: ");
          leido = readLineTimeout(br, 10000);
      } catch (TimeoutException | IOException e) {
          antesDe10=false;

      }

      System.out.println( antesDe10?leido:fueraDeTiempo);
  }
 }
```

SALIDA ESTÁNDARD

Hay dos salidas standard:

- La salida de datos "normal": System.out.
- La salida de errores: System.err

Sobre System.out

System.out, referencia a un objeto PrintStream. La clase PrintStream tiene los famosisimos métodos print() y println() por lo que ya que System.out es de tipo PrintStream puede utilizar dichos métodos.

La referencia System.out podemos copiarla como otra referencia cualquiera

```
import java.io.PrintStream;
class App{
    public static void main(String[] args){
        System.out.println("Hola");
        PrintStream ps=System.out;
        ps.println("adios");
    }
}
```