élogos

OPERACIONES DE ENTRADA-SALIDA

élogos

ÍNDICE

OPERACIONES DE ENTRADA-SALIDA

1. La clase PrintStream para salida de datos

Clase PrintStream

La clase PrintStream tiene métodos para la representación textual de todos los datos primitivos de Java: write y println(). PrintStream es la clase universal con la que podemos enviar información a cualquier dispositivo o stream externo.

La expresión System.out hace referencia al objeto PrintStream asociado a la salida estándar, es decir, la consola MS-DOS.

Ejemplo:

```
System.out.println ("Hola desde Java");
int x=3;
System.out.println (x);
```

2. Lectura de caracteres con InputStream

Streams

Un "Stream" es un nombre genérico otorgado a un flujo de caracteres. Puede estar compuesto por los valores residentes en un archivo de texto, datos introducidos interactivamente por un usuario o datos que deseen ser colocados en un determinado archivo.

Stream es una secuencia de bytes que viajan desde un origen (source) hasta un destino (target). El concepto de Stream se aplica a multitud de dispositivos de entrada/salida.

Para encapsular los Streams de Java se tienen dos clases:

- InputStream (entran en la memoria).
- OutputStream (salen de memoria a otro dispositivo).

InputStream

Entre los principales métodos de esta clase tenemos:

abstract int read() throws IOException:

Lee un solo byte del Stream, retorna un entero con valor comprendido entre 0…255 ó –1 para el final del Stream.

int read(byte[]b)

int read(byte []b, int offset, int length):

Ambos métodos realizan la lectura de una serie de bytes. En el primer prototipo se realiza la lectura del array de bytes y en el segundo se realiza la lectura de un array desde la posición indicada en offset y el número de bytes indicado en length.

int available():

Devuelve el número de bytes que se puede leer del Stream sin quedar bloqueado.

void close():

Cierra el Stream.

boolean markSupported():

Devuelve true si el Stream soporta marcas. Por defecto, devuelve false.

void mark (int readlimit):

Pone una marca en el Stream para poder situarse en este punto tras haberse movido hacia adelante. El parámetro readlimit indica el número máximo de bytes que la marca es válida, esto se debe al consumo de memoria que hay que reservar.

void reset():

Vuelve a la marca.

long skip (long n) throws IOException:

Avanza hacia delante el número de bytes que se le indique en n. Como retorno devuelve el número de bytes que se ha avanzado.

OutputStream

Representa el stream de salida. Entre sus principales métodos destacar:

abstract void write (int b):

Escribe un solo byte representado por b.

void write (byte []b)

void write (byte []b, int offset, int length):

Reciben un array para escribir varios bytes y no sólo uno.

void flush()

Se utilizan en los Streams con buffer (trozo de memoria de la RAM) para sacar lo almacenado en el buffer.

void close()

Se cierra el Stream.

La clase FileInputStream

La clase FileInputStream es una subclase de InputStream que permite recuperar datos desde un fichero. Se puede crear un objeto de esta clase a partir de la ruta del fichero que se va a tratar.

Los constructores son los siguientes:

FileInputStream (String Filename) . Crea el objeto a partir de la ruta del fichero

FileInputStream (File Filename). Crea el objeto a partir del objeto File asociado al fichero

La clase FileOutputStream

La clase FileOutputStream también sirve para acceder al contenido de los ficheros. Realiza la escritura de los datos del fichero.

Los constructores son los siguientes:

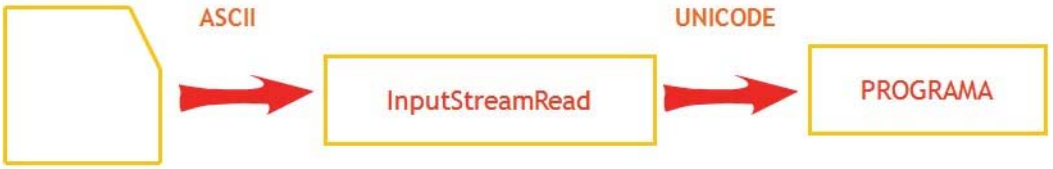
FileOutputStream (String Filename) throws IOException. Con este constructor se empieza desde el principio en el fichero, aunque el fichero tenga datos.

FileOutputStream(String Filename, boolean append). Con este constructor se añadirá información en el fichero. Para ello se pondrá a true la variable append. Si se pone false funcionará como el anterior constructor.

3. Lectura de cadenas con BufferedReader

InputStreamReader

La clase InputStreamReader permite la conversión de código ASCII a UNICODE, actuando como filtro para la traducción.



El problema surge en cómo traducir de ASCII a UNICODE, los caracteres nacionales (0..127 Mundiales / 128..255 Nacionales). Este problema se resuelve con el siguiente constructor:

InputStreamReader (InputStream is, String encode)

El segundo parámetro representa el código ISO que se quiere utilizar. En el caso de España es "8859-1".

La InputStreamReader lee caracteres desde teclado, pero procesa carácter a carácter, no sirve para leer de golpe una línea.

System.in.read

Para leer caracteres desde teclado, también usamos la clase System.in.read(); Esta función nos devuelve el ASCII del carácter pulsado desde el teclado, y lo hace como tipo int.

BufferedReader

La clases BufferedReader nos sirven para leer por teclado, sin procesar carácter a carácter.

La manera de crear un objeto de BufferedReader a partir de otro Reader cualquiera, como por ejemplo, InputStreamReader, es el siguiente:

BufferedReader br = new BufferedReader (new InputStreamReader(System.in));

El funcionamiento de esta clase es igual que el InputStreamReader, solamente que nos va a devolver la cadena leída completa hasta que pulsamos > (intro)

Para obtener el String leido se usa el método readLine(). Este método lee todos los caracteres tecleados (recibidos si fuera otro dispositivo de entrada) hasta que encuentra la pulsación de la tecla <INTRO> >

String cadena = br.readLine();

Lee desde el teclado un String completo y lo guarda en la variable "cadena".

4. Lectura de datos con la clase Scanner

Clase Scanner

La clase Scanner, incorporada a partir de la versión Java 5, permite realizar la lectura de datos en un programa de forma muy sencilla.

Ejemplo:

```
import java.io.*;

import java.util.Scanner;

public class Leer{

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

{

Scanner leer=new Scanner(System.in); //Creamos el objeto de Scanner

String cadena=leer.next(); // Leemos el texto con su método next()

System.out.println  (“Texto leido :” + cadena ); //Lo imprimimos

}

}
```

Configurar clase Scanner

Se puede configurar la clase Scanner para que utilice cualquier otro tipo de separador de tokens. Para ello, utilizaremos el método useDelimiter() proporcionado por la clase

Además tiene una serie de métodos que tras leer un token lo intentan interpretar como algún tipo primitivo de Java:

int nextInt()	double nextDouble()
byte nextByte()	float nextFloat()
short nextShort()	boolean nextBoolean()
long nextLong()	

Cuando analizamos valores numéricos se emplean los convenios locales de puntuación. Scanner se usa a menudo para la entrada de datos por consola:

```
Scanner scanner = new Scanner(System.in);

System.out.print("Escribe dos números: ");

double x = scanner.nextDouble();

double y = scanner.nextDouble();

System.out.println("producto: " + (x * y));
```