11 - Entrada y Salida de datos

Programació Orientada a Objectes

Mario Macías

OutputStream (flujo de salida)

- Clase **abstracta** que provee métodos básicos para la salida de datos
  - Todas sus subclases se utilizarán de la misma manera
- Métodos más comunes
  - void write(int b)
    - Escribe el byte más bajo (valor 0 a 255) contenido en el int
  - void write(byte[] buffer)
    - Escribe un conjunto de bytes contenidos en el buffer
  - void close()
    - Cierra el flujo de datos, y libera los recursos asociados a éste
    - Es **muy importante** llamar a este método cuando no vayamos a utilizar más un *stream*, ya que si no los recursos podrían quedarse bloqueados por un tiempo durante el cual no se podrían volver a usar
  - Los métodos anteriores pueden lanzar una IOException si sucede algún error de E/S

PrintStream

- Subclase de OutputStream
- Simplifica el envío de cadenas de texto a otro OutputStream o a disco:
  - Algunos constructores:
    - public PrintStream(OutputStream streamSalida)
    - public PrintStream(String nombreFichero)
  - System.out (print, println, flush…) es un PrintStream que envía texto a la pantalla

Receta

```
try {

  PrintStream disco
               = new PrintStream("datos.txt");

  disco.println("En un lugar");
  disco.println("de La Mancha...");
  disco.println("\nCervantes");

  disco.close();

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

InputStream (flujo de entrada)

- Clase **abstracta** que provee métodos básicos para la entrada de datos
  - Todas sus subclases se utilizarán de la misma manera
- Métodos más comunes
  - int read()
    - Lee un byte (valor 0 a 255) y lo retorna como int.
    - Retorna -1 si se ha llegado al final del flujo
  - int read(byte[] buffer)
    - Lee un conjunto de bytes y los guarda en el parámetro 'buffer'
    - Retorna el número de bytes leídos, o -1 si se ha llegado al final del flujo
  - void close()
    - Cierra el flujo de datos, y libera los recursos asociados a éste
    - Es **muy importante** llamar a este método cuando no vayamos a utilizar más un *stream*, ya que si no los recursos podrían quedarse bloqueados por un tiempo durante el cual no se podrían volver a usar
  - Los métodos anteriores pueden lanzar una IOException si sucede algún error de E/S

FileInputStream

- Subclase de InputStream que lee datos de disco
- Constructor: public FileInputStream(String nombreFichero)
  - Lanza FileNotFoundException si no se encuentra el archivo

Scanner

- La clase Scanner simplifica la lectura de texto de un InputStream

- Constructor:

```
public Scanner(InputStream streamALeer)
```

- Ejemplo:

```
// System.in es un InputStream que lee
// datos del teclado
Scanner scanner = new Scanner(System.in);
String lineaLeida = scanner.nextLine();
```

Receta

```
try {
  Scanner scanner = new Scanner(
       new FileInputStream("datos.txt"));
  while(scanner.hasNextLine()) {
    String linea = scanner.nextLine();
    System.out.println(linea);
  }
  scanner.close();
} catch(FileNotFoundException ex) {
  ex.printStackTrace();
}
```