EJERCICIOS PROGRAMACIÓN MULTIPROCESO – JAVA

⇒ **Los ejercicios a realizar en clase son: Ejemplo 2, Ejemplo 3, Ejemplo 4, EjemploLectura y Ejemplo 5. No se entregan en Aules.**
⇒ **Los ejercicios a entregar en Aules son: Actividad 1.4 y Actividad 1.6.**

El siguiente ejemplo ejecuta el comando DIR. Usaremos el método **getInputStream()** de la clase **Proccess** para leer el stream de salida del proceso, es decir, para leer lo que el comando DIR envía a la consola. Definiremos así el stream:

```
InputStream is = p.getInputStream();
```

Para leer la salida usamos el método **read()** de **InputStream** que nos devolverá carácter a carácter la salida generada por el comando. El programa Java es el siguiente:

```
import java.io.*;
public class Ejemplo2 {
  public static void main(String[] args) throws IOException {

     //Ejecutamos el proceso DIR
     Process p = new ProcessBuilder("CMD", "/C", "DIR").start();

     //Mostramos carácter a carácter la salida generada por DIR
     try {
        InputStream is = p.getInputStream();
        int c;
        while ((c = is.read()) != -1)
             System.out.print((char) c);
        is.close();

        } catch (Exception e) {
             e.printStackTrace();
        }

        //COMPROBACIÓN DE ERROR - 0 bien - 1 mal

     int exitVal;
     try {
          exitVal = p.waitFor(); //recoge la salida de System.exit()
          System.out.println("Valor de Salida: " + exitVal);
     } catch (InterruptedException e) {
          e.printStackTrace();
     }
  }
}// Ejemplo2
```

Al ejecutarlo, desde el entorno Eclipse, se muestra una salida similar a la siguiente:

```
El volumen de la unidad D es Data
El número de serie del volumen es: 9013-7A66

Directorio de D:\CLASE\PSP_2018\CAPITULO1

14/06/2018  00:14    <DIR>          .
14/06/2018  00:14    <DIR>          ..
14/06/2018  00:14               396 .classpath
14/06/2018  00:14               385 .project
14/06/2018  00:14    <DIR>          .settings
14/06/2018  00:15    <DIR>          bin
14/06/2018  00:15    <DIR>          src
               2 archivos            781 bytes
               5 dirs  134.146.781.184 bytes libres
Valor de Salida: 0
```

El método **waitFor()** hace que el proceso actual espere hasta que el subproceso representado por el objeto **Process** finalice. Este método recoge lo que **System.exit()** devuelve, por defecto en un programa Java si no se incluye esta orden el valor devuelto es 0, que normalmente responde a una finalización correcta del proceso.

El siguiente ejemplo muestra un programa Java que ejecuta el programa Java anterior, en este caso el programa se ejecutará desde el entorno Eclipse. Como el proceso a ejecutar se encuentra en la carpeta **bin** del proyecto será necesario crear un objeto **File** que referencie a dicho directorio. Después para establecer el directorio de trabajo para el proceso que se va a ejecutar se debe usar el método **directory()**, a continuación se ejecutará el proceso y por último será necesario recoger el resultado de salida usando el método **getInputStream()** del proceso:

```
import java.io.*;
public class Ejemplo3 {
  public static void main(String[] args) throws IOException {

    //creamos objeto File al directorio donde esta Ejemplo2
    File directorio = new File(".\\bin");

    //El proceso a ejecutar es Ejemplo2
    ProcessBuilder pb = new ProcessBuilder("java", "Ejemplo2");

    //se establece el directorio donde se encuentra el ejecutable
    pb.directory(directorio);

    System.out.printf("Directorio de trabajo: %s%n",pb.directory());

    //se ejecuta el proceso
    Process p = pb.start();

    //obtener la salida devuelta por el proceso
    try {
          InputStream is = p.getInputStream();
          int c;
          while ((c = is.read()) != -1)
                System.out.print((char) c);
          is.close();
    } catch (Exception e) {
          e.printStackTrace();
    }
  }
}// Ejemplo3
```

La salida mostrará los ficheros y carpetas del directorio definido en la variable *directorio*. Si ambos ficheros están en la misma carpeta o directorio, no será necesario establecer el directorio de trabajo para el objeto **ProcessBuilder**. Si el *Ejemplo2* a ejecutar se encontrase en la carpeta D:\PSP, tendríamos que definir el objeto *directorio* de la siguiente manera: *File directorio = new File("D:\\PSP").*

ACTIVIDAD 1.4

Crea un programa Java llamado *LeerNombre.java* que reciba desde los argumentos de *main()* un nombre y lo visualice en pantalla. Utiliza *System.exit(1)* para una finalización correcta del programa y *System.exit(-1)* para el caso que no se hayan introducido los argumentos correctos en *main()*.

A continuación, haz un programa parecido a *Ejemplo3.java* para ejecutar *LeerNombre.java*. Utiliza el método **waitFor()** para comprobar el valor de salida del proceso que se ejecuta. Prueba la ejecución del programa dando valor a los argumentos de *main()* y sin darle valor. ¿Qué valor devuelve **waitFor()** en un caso y en otro?

ENVIAR DATOS AL STREAM DE ENTRADA DEL PROCESO

Supongamos ahora que queremos ejecutar un proceso que necesita información de entrada. Por ejemplo, si ejecutamos DATE desde la línea de comandos y pulsamos la tecla [Intro] nos pide escribir una nueva fecha:

```
D:\CAPIT1>DATE
La fecha actual es: 14/06/2018
Escriba la nueva fecha: (dd-mm-aa) 15-06-18
```

La clase **Process** posee el método **getOutputStream()** que nos permite escribir en el stream de entrada del proceso, así podemos enviarle datos. El siguiente ejemplo ejecuta el comando DATE y le da los valores 15-06-18. Con el método **write()** se envían los bytes al stream, el método **getBytes ()** codifica la cadena en una secuencia de bytes que utilizan juego de caracteres por defecto de la plataforma:

```
import java.io.*;
public class Ejemplo4 {
  public static void main(String[] args) throws IOException {

      Process p = new ProcessBuilder("CMD", "/C", "DATE").start();

      // escritura -- envia entrada a DATE
      OutputStream os = p.getOutputStream();
      os.write("15-06-18".getBytes());
      os.flush(); // vacía el buffer de salida

      // lectura -- obtiene la salida de DATE
      InputStream is = p.getInputStream();
      int c;
      while ((c = is.read()) != -1)
           System.out.print((char) c);
      is.close();

      // COMPROBACION DE ERROR - 0 bien - 1 mal
      int exitVal;
      try {
            exitVal = p.waitFor();
            System.out.println("Valor de Salida: " + exitVal);
      } catch (InterruptedException e) {
            e.printStackTrace();
      }
  }
```

La ejecución muestra la siguiente salida:

```
La fecha actual es: 14/06/2018
Escriba la nueva fecha: (dd-mm-aa) 15-06-18
Valor de Salida: 0
```

Supongamos que tenemos un programa Java que lee una cadena desde la entrada estándar y la visualiza:

```
import java.io.*;
public class EjemploLectura{
 public static void main (String [] args)
 {
   InputStreamReader in = new InputStreamReader(System.in);
   BufferedReader br = new BufferedReader (in);
   String texto;
   try {
    System.out.println("Introduce una cadena....");
    texto= br.readLine();
    System.out.println("Cadena escrita: "+texto);
    in.close();
   }catch (Exception e) { e.printStackTrace();}
 }
}//EjemploLectura
```

Con el método **getOutputStream()** podemos enviar datos a la entrada estándar del programa *EjemploLectura.java*. Por ejemplo si queremos enviar la cadena "Hola Manuel" cambiaríamos varias cosas en el *Ejemplo4.java*:

```
File directorio = new File(".\\bin");
ProcessBuilder pb = new ProcessBuilder("java", "EjemploLectura");
pb.directory(directorio);

// se ejecuta el proceso
Process p = pb.start();

// escritura - se envia la entrada
OutputStream os = p.getOutputStream();
os.write("Hola Manuel\n".getBytes());
os.flush(); // vacía el buffer de salida
```

Cada línea que mandemos a *EjemploLectura* debe terminar con "\n", igual que cuando escribimos desde el terminal la lectura termina cuando pulsamos la tecla [Intro]. Suponiendo que hemos guardado estos cambios en *Ejemplo5.java*, la ejecución muestra la siguiente salida:

```
Introduce una cadena....
Cadena escrita: Hola Manuel
Valor de Salida: 0
```

ACTIVIDAD 1.6

Escribe un programa Java que lea dos números desde la entrada estándar y visualice su suma. Controlar que lo introducido por teclado sean dos números. Haz otro programa Java para ejecutar el anterior. Realiza el ejercicio 7.
