

The background features a dark blue field with intricate white and light blue circuit-like lines. Several glowing green circles of varying sizes are scattered throughout. On the left side, there are three interlocking gears of different sizes, rendered in a light blue, semi-transparent style. The main title is centered in a large, bold, green font.

CONCURRENT PROGRAMMING IN JAVA

UNIT 1: PROCESS PROGRAMMING



CONCURRENCIA DE PROCESOS EN JAVA

- La funcionalidad básica para *procesos* la proporciona la clase **Process**.
- La clase **Process** es abstracta y, por lo tanto, no se pueden crear objetos de ella, sino de subclases suyas.
- La clase **Process** tiene métodos para lanzar un proceso, obtener información acerca de su estado y controlar su ejecución.
- Se pueden crear objetos de la clase **Process** con métodos de otras clases:
 - **ProcessBuilder** usando el método **start()**
 - **Runtime** usando el método **exec()**.
- Cada programa de Java tiene asociada una instancia de la clase **Runtime** que le permite obtener información de su entorno de ejecución e interactuar con él. Este se puede recuperar con el método estático **getRuntime()**.
- Las diversas variantes del método **exec()** permiten lanzar nuevos procesos.



MÉTODOS DE LA CLASE RUNTIME

Método	Funcionalidad
<code>static Runtime getRuntime()</code>	Devuelve el objeto Runtime asociado con la aplicación de Java actual.
<code>Process exec (String command)</code> <code>Process exec (String[] comandoYArgs)</code> <code>Process exec (String[] comandoYArgs, String[] envp)</code> <code>Process exec (String[] cmdarray, String[] envp, File dir)</code>	Ejecuta un comando. Algunas variantes permiten especificar el entorno de ejecución y el directorio para ejecutar el comando.
<code>void exit (int status)</code> <code>void halt (int status)</code>	Termina la ejecución de la máquina virtual. exit lo hace de manera ordenada. halt lo hace de manera abrupta.
<code>int availableProcessors()</code>	Devuelve el número de procesadores disponibles para la máquina virtual de Java.
<code>long freeMemory()</code>	Devuelve la cantidad de memoria disponible para la máquina virtual.



MÉTODOS DE LA CLASE PROCESSBUILDER

Método	Funcionalidad
ProcessBuilder (String... comando) ProcessBuilder (List<String> comando)	Construye un ProcessBuilder para un comando con los correspondientes argumentos de línea de comandos, en su caso.
ProcessBuilder directory (File directorio) File directory()	El primer método establece el directorio de trabajo del proceso, y el segundo método lo devuelve. Por defecto, el segundo método devuelve null. Si es así, se puede obtener el directorio de ejecución del proceso actual con System.getProperty("user.dir") .
Map<String, String> environment()	Devuelve el entorno de trabajo del proceso, que es una lista de asignación de valores para variables de entorno.
Process start()	Crea e inicia un proceso.



MÉTODOS DE LA CLASE PROCESS (I)

Método	Funcionalidad
<code>void destroy()</code> <code>public Process destroyForcibly()</code>	Termina el proceso. El primer método permite una terminación limpia y ordenada del proceso. El segundo lo termina inmediatamente.
<code>int exitValue()</code>	Devuelve el valor de salida, o código de retorno, del proceso. Por convención, un valor 0 indica terminación normal , y cualquier otro valor se interpretará como un código de error . Se puede terminar un programa en Java con un código de retorno distinto de 0 con <code>System.exit(código)</code> .
<code>ProcessHandle.Info info()</code>	Devuelve la información actual del proceso.
<code>boolean isAlive()</code>	Comprueba si el proceso está vivo.
<code>long pid()</code>	Devuelve el PID o identificador de proceso.



MÉTODOS DE LA CLASE PROCESS (II)

Método	Funcionalidad
<code>int waitFor()</code>	<p>Hace que el hilo en ejecución espere hasta que el proceso haya terminado. Devuelve el valor de salida del proceso.</p> <p>Un valor 0 se entiende que corresponde a una ejecución sin errores.</p> <p>Un valor distinto de cero corresponde a un código de error.</p> <p>Si el proceso es de un programa en Java, es el valor devuelto por System.exit(), o cero si no se terminó la ejecución con System.exit().</p>
<code>Boolean waitFor (long timeout, TimeUnit unit)</code>	<p>Hace que el hilo en ejecución espere hasta que el proceso haya terminado, durante un tiempo máximo indicado por timeout.</p> <p>TimeUnit proporciona distintas constantes para indicar la unidad de medida: horas, días, milisegundos, etc.</p> <p>Devuelve true si el proceso ha terminado por sí mismo antes del tiempo máximo indicado, y false en caso contrario.</p>



TOMA NOTA

El siguiente fragmento de código no funciona:

```
ProcessBuilder pb = new ProcessBuilder("df -h /");  
pb.start();
```

Produce la siguiente excepción:

```
java.io.IOException: Cannot run program "df - h /": error=2, No existe el  
archivo o el directorio
```

Esto es porque **ProcessBuilder** tiene constructores para una lista de String o para varios String, e interpreta el primer String como el programa que ejecutar, y el resto, como parámetros de línea de comandos. Si solo se le pasa "df -h /", intenta ejecutar un programa "df -h /", que no existe, lo que sí existe es el programa df.

Para que funcione tendríamos que usar alguno de los siguientes constructores:

```
ProcessBuilder pb = new ProcessBuilder("df", "-h", "/");  
ProcessBuilder pb = new ProcessBuilder(new String[] {"df", "-h", "/"});
```



REDIRECCIÓN DE E/S ESTÁNDARES Y DE ERROR

Las clases `ProcessBuilder` y `Process` proporcionan diversos mecanismos para redireccionar y gestionar la salida y entrada de los procesos.

- **Entrada estándar (*standard input*):** Es un stream de entrada del que puede leer datos. Por defecto, suele estar asociada al teclado.
- **Salida estándar (*standard output*):** Es un stream de salida en el que se pueden escribir datos. Por defecto, suele estar asociada a la pantalla.
- **Salida de error (*error output*):** Es un stream de salida en el que se pueden escribir datos. También suele estar asociada a la pantalla.




MECANISMOS DE REDIRECCIÓN DE PROCESSBUILDER

La clase **ProcessBuilder** proporciona mecanismos para la redirección de la entrada y salida de los nuevos procesos creados.

En el primer ejemplo visto en clase usaba el método **inheritIO()** para enlazar la entrada y salida estándares y de error de los subprocesos con las del proceso principal, y por eso podíamos ver su salida.



REDIRECCIÓN DE ENTRADA Y SALIDA CON PROCESSBUILDER

Método	Funcionalidad
<code>ProcessBuilder inheritIO()</code>	Redirige la salida estándar y de error de los subprocesos creados hacia las del proceso padre, y su entrada estándar desde la del proceso padre.
<code>ProcessBuilder redirectInput(File f)</code> <code>ProcessBuilder redirectOutput(File f)</code> <code>ProcessBuilder redirectError(File f)</code>	Estos métodos redirigen la entrada y la salida estándares y de error desde o hacia un fichero.
<code>ProcessBuilder redirectInput(ProcessBuilder.Redirect fuente)</code> <code>ProcessBuilder redirectOutput(ProcessBuilder.Redirect destino)</code> <code>ProcessBuilder redirectError(ProcessBuilder.Redirect destino)</code>	<p>Estos métodos redirigen la entrada y salida estándares y de error desde o hacia la correspondiente del proceso padre con Redirect.INHERIT.</p> <p>La salida estándar y de error se pueden descartar con Redirect.DISCARD.</p> <p>Con Redirect.appendTo(File file) añadimos contenido a un fichero y no sobrescribimos el contenido</p> 

REDIRECCIÓN DE ENTRADA Y SALIDA CON PROCESSBUILDER (II)

Método	Funcionalidad
<code>ProcessBuilder redirectErrorStream (boolean redir)</code>	Con redir=true dirige la salida de error hacia la estándar.
<code>static List<Process> startPipeline (List<ProcessBuilder> builders)</code>	Inicia un proceso para cada ProcessBuilder, y crea una secuencia de procesos cuyas salida y entradas están enlazadas mediante tuberías o pipes , de manera que la salida de un proceso se dirige a la entrada del siguiente en la secuencia. Solo se pueden redirigir la entrada del primer proceso y la salida del último. El resto no se pueden redirigir.



REDIRECCIÓN DE ENTRADA Y SALIDA CON PROCESS

Método	Funcionalidad
<code>OutputStream getOutputStream()</code>	Devuelve un <i>stream</i> de salida conectado con la entrada estándar del proceso.
<code>InputStream getInputStream()</code>	Devuelve un <i>stream</i> de entrada conectado con la salida estándar del proceso.
<code>InputStream getErrorStream()</code>	Devuelve un <i>stream</i> de entrada conectado con la salida estándar de error del proceso.

