

PSP - Examen 1a Evaluación

Pipeline de procesos

Proyecto base

Las clases proporcionadas y los archivos de prueba no se pueden modificar.

En la entrega del ejercicio no se puede usar el método `inheritIO()`.

Clase principal: psp.examenes.MainPipeline

Vas a implementar un programa **orquestador** que ejecuta **dos procesos Java** en cadena:

- ProcesadorTexto (primer proceso)
- ResumenTexto (segundo proceso)

Las clases correspondientes a estos dos procesos ya se proporcionan implementadas.

El programa principal debe llamarse `MainPipeline` y se ejecuta como aplicación de consola recibiendo parámetros en el `main`.

Parámetros del programa principal

El programa se ejecutará así:

```
java MainPipeline <input.txt> <out1.txt> <out2.txt> <modo> <timeoutSeg>
```

- `input.txt` : fichero de entrada (texto)
- `out1.txt` : fichero donde se guardará la salida del primer proceso
- `out2.txt` : fichero donde se guardará la salida del segundo proceso
- `modo` : U (mayúsculas) o L (minúsculas)
- `timeoutSeg` : entero > 0 (segundos)

salida por consola

Si faltan parámetros o son inválidos, el programa debe mostrar **un mensaje de uso** y finalizar con **código ≠ 0**.

`MainPipeline`, tras recibir y procesar los parámetros, debe realizar los siguientes pasos:

Parte A — Lanzar el primer proceso con argumentos y redirecciones

1. Lanza `ProcesadorTexto`, pasándole el fichero `input.txt` **como redirección de entrada** (es decir: el proceso leerá el contenido de `input.txt` desde `STDIN`).
2. La salida estándar (`STDOUT`) del proceso 1 debe guardarse en `out1.txt` mediante la **redirección de la salida** del proceso.
3. Los errores (`STDERR`) del proceso 1 deben **redirigirse** a un fichero `errores1.log`.
4. Tras lanzar el proceso 1, `MainPipeline`, debe esperar su finalización un tiempo límite recibido como parámetro. Si supera el timeout debe terminar con código 3.

Parte B — Lanzar el segundo proceso gestionando su E/S

1. Solo si el proceso anterior termina con código 0, se debe lanzar el segundo proceso `ResumenTexto`.
2. Leer el contenido del archivo `out1.txt` y **enviarlo, línea a línea**, al `STDIN` del proceso `ResumenTexto`.
3. Captura la salida de `ResumenTexto` (sin `inheritIO`) y **escribirla directamente** en `out2.txt`, pero no con `redirectOutput`.
Debe hacerse leyendo `process2.getInputStream()` y escribiendo a `FileOutputStream(out2.txt)`.
4. Los errores del proceso se deben **redirigir** a `errores2.log`.

☰ Clases ya implementadas en el proyecto base

Las clases `ProcesadorTexto` y `ResumenTexto` no hay que implementarlas. Se proporcionan como parte del proyecto base del ejercicio.

Proceso 1: psp.examenes.ProcesadorTexto

Debe ejecutarse como un programa Java independiente:

```
java ProcesadorTexto <modo>
```

- **Entrada:** por **STDIN** recibe texto (líneas).
- **Salida:** por **STDOUT** escribe el texto transformado:
 - Si `modo` = U: convierte a mayúsculas
 - Si `modo` = L: convierte a minúsculas

Errores:

Si modo no es válido, debe escribir un error por **STDERR** y salir con código 2.

⚠ E/S del proceso

`ProcesadorTexto` **NO** debe leer archivos directamente. Solo STDIN/STDOUT.

Proceso 2: psp.examenes.ResumenTexto

Debe ejecutarse como un programa Java independiente:

```
java ResumenTexto
```

- **Entrada:** por STDIN recibe texto (líneas).
- **Salida:** por STDOUT imprime un resumen con este formato:

```
LINEAS=<n>
PALABRAS=<m>
CARACTERES=<k>
```

donde

- LINEAS: número de líneas de la entrada
- PALABRAS: número total de palabras (separadas por uno o más espacios)
- CARACTERES: total de caracteres **incluyendo espacios** (pero no hace falta contar saltos de línea si no quieres; documenta tu criterio).

⚠ E/S del proceso

`ResumenTexto` NO lee archivos directamente.