PSP - U2 Actividades

Descargar estos apuntes

Índice

LectorDialogos

> ⚠ **Proyecto base**
>
> El proyecto base LectorDialogos contiene la clase LectorFicheros que se usará como proceso hijo. **No se puede modificar nada en esa clase**.
> En la entrega del ejercicio no se puede usar el método `inheritIO()`.

psp.examenes.LectorDialogos

Cread esta clase para que tenga la funcionalidad que se describe a continuación:

- La clase recibirá un **parámetro de entrada** con el nombre de alguno de los archivos (sólo el nombre, sin especificar la ruta) que hay en la carpeta `datos` del proyecto.
- Esta clase actuará como **proceso principal**. Se encargará de lanzar, en primer lugar, la clase `LectorFicheros` como proceso hijo.
- El proceso hijo tendrá **redirigida su entrada estándar** al fichero que LectorDiálogos recibido como parámetro.
- El proceso padre esperará a que finalice el proceso hijo, y mostrará el valor de finalización del mismo.
- A continuación, recuperará la salida estándar del proceso hijo y la mostrará, también por la salida estándar, línea a línea, precedida del texto `P1`.
- Por último, recuperará la salida de error del proceso hijo y la mostrará, también por la salida de error, línea a línea, precedida del texto `P2>`.

Una vez finalice el proceso anterior, el proceso principal lanzará otro proceso con las siguientes características:

- Ejecutar el comando de consola `sort`
- Redirigir su entrada estándar desde el archivo “datos/listaAlumnado.txt”.
- De este proceso recogemos la salida estándar para mostrarla por consola, precediendo cada línea con el prefijo `ordenarPalabras>`.
- De este proceso recogemos la salida de error para mostrarla por consola (por la salida de error), precediendo cada línea con el prefijo `cmdError>`

Cuando finalice la ejecución del proceso deberemos recuperar su código de finalización y mostrarlo por consola.

Estación meteorológica

Queremos crear una aplicación para simular el funcionamiento de una estación meteorológica y los datos que se van generando en los sensores que incorpora.

Dentro de la estación, tenemos varios sensores que van generando información cada cierto tiempo.
Cuando encendemos la estación, hay un `programa principal` que se encarga de lanzar cada uno de los `drivers` asociados a los sensores manteniendo la comunicación con ellos y recibiendo la información que los sensores van generando secuencialmente.

En la estación meteorológica tenemos sensores para medir:

- Temperatura: Genera un valor de temperatura entre **-10 y 40 grados centígrados cada 2 segundos**.
- Humedad: Genera un valor de humedad entre **0% y 100% cada 3 segundos**.
- Presión atmosférica: Genera un valor de presión entre **950 y 1050 hPa cada 5 segundos**.

Se pide realizar una aplicación en Java que simule el funcionamiento de la estación meteorológica y todos los `drivers` asociados, mostrando por consola los mensajes necesarios para entender qué está pasando en cada momento.

Cada `driver de un sensor` lo vamos a simular con un **proceso**.

- Los drivers de los sensores deben funcionar de forma independiente, generando la información de forma asíncrona.
- Cada driver está parametrizado para recibir como `argumentos` el nombre del Sensor, los valores mínimos y máximos que puede generar, las unidades de medida y el tiempo que tarda en capturar cada valor.

La estación hace un polling (lectura periódica) **cada 10 segundos** a cada `driver de sensor` para recoger la información que va generando cada sensor.

- En ese tiempo, cada sensor puede haber generado varios valores.
- La estación debe recoger todos los valores generados por cada sensor desde la última lectura y mostrarlos por consola.

> **Información del proceso**
>
> La aplicación debe mostrar claramente qué está pasando en cada momento.
>
> La salida por consola debe permitir distinguir fácilmente qué sensor genera qué dato, el dato en sí y cuándo se ha generado `(colores opcionales, nombres obligatorios)`.
> *El nombre de los procesos debe establecerse como "SensorTemperatura", "SensorHumedad", "SensorPresion", etc.*

> **Errores en los sensores**
>
> Los sensores que hemos comprado **fallan cada 10 lecturas** (tienen un 10% de probabilidad de fallar), generando un error en la lectura.
>
> En caso de error, el `driver del sensor` debe enviar un mensaje de error al programa principal en lugar del valor medido.
>
> El programa principal debe diferenciar y manejar estos errores adecuadamente, mostrando un mensaje de error por consola que especifique qué sensor ha fallado y en qué momento.

Procesos genéricos

En cualquier momento puede añadirse un nuevo sensor a la estación.
Programa el programa principal y los drivers de forma que pueda lanzar cualquier número de procesos, leyendo sus parámetros desde una colección de datos que incluya, al menos, la siguiente información:

```
# nombre_sensor,valor_minimo,valor_maximo,unidades,tiempo_lectura
SensorTemperatura,-10,40,grados,2000
SensorHumedad,0,100,porcentaje,3000
SensorPresion,950,1050,hPa,5000
```

Carga inicial de datos

La colección de datos está cargada en memoria cuando se inicia el programa principal.
Tienes que encargarte de crear y rellenar esa colección de datos.

Simulación en escala

Se pide simular en escala el proceso de cada sensor.
Cada segundo real = 100 ms en la simulación.

Gestión de excepciones en el código

Controla y maneja excepciones en tu código (no las escales), mostrando la pila cuando sea necesario.

Rúbrica de corrección

Criterio	Ponderación	Excelente (5)	Bien (3)	Mal (1)	Insuficiente (0)
Diseño y organización del código		Código modular, limpio y bien estructurado, con separación clara de procesos y comunicación.	Código organizado, con ligeras mejoras posibles en modularidad y claridad.	Código funcional pero con estructura y modularidad limitadas.	Código desorganizado, difícil de seguir.
Uso de ProcessBuilder y Process		Uso correcto y completo de ProcessBuilder y Process para crear y manejar procesos externos.	Uso mayormente correcto con algún manejo incompleto o subóptimo.	Uso parcial o incompleto de las clases para procesos.	Uso incorrecto o inexistente de procesos externos.
Comunicación a través de Streams		Comunicación clara y correcta mediante InputStream y OutputStream con protocolo definido y eficiente.	Comunicación efectiva, pero con pequeños aspectos mejorables en protocolo o claridad.	Comunicación funcional pero poco robusta o clara.	Comunicación inexistente o mal implementada.
Manejo de errores y excepciones		Maneja correctamente excepciones en creación y comunicación con procesos, con mensajes claros.	Manejo adecuado pero con faltas en algunos casos o mensajes deficientes.	Manejo limitado y poco consistente de errores.	No hay manejo o es deficiente, generando caídas o bloqueos.
Control de flujo sin sincronización		Diseña un control de flujo eficaz que evita pérdidas o sobrescrituras con comunicación sin sincronización.	Control de flujo adecuado con mínimos aspectos mejorables.	Control de flujo básico, con posibles inconsistencias.	No controla flujo correctamente, errores frecuentes.
Salida y estado del sistema		Salidas claras, informativas y detalladas que reflejan el estado de cada proceso y la comunicación.	Salidas completas pero quizás menos detalladas o formateadas.	Salidas funcionales pero poco claras o con poca información.	Salidas confusas, inexistentes o poco útiles para entender el sistema.
Parámetros y configuración		Parámetros completamente	Parámetros configurables pero	Algunos parámetros	Sin parámetros configurables ni

Criterio	Ponderación	Excelente (5)	Bien (3)	Mal (1)	Insuficiente (0)
		configurables y documentados, facilidad para variar pruebas.	con poca documentación o flexibilidad limitada.	ajustables, poca documentación.	documentación sobre ellos.
Documentación y explicaciones		Explicación clara y detallada del diseño, protocolo y estrategia de comunicación y control.	Documentación adecuada con áreas mejorables en detalles.	Documentación básica con falta de profundidad o claridad.	Documentación insuficiente o inexistente.