**Ejercicios unidad 1 y 2**

Contenido

[1 Procesos 2](#_Toc149245874)

[2 Fichero, escritura simultanea 3](#_Toc149245875)

[3 Pajaros 4](#_Toc149245876)

[3.1 Enunciado 4](#_Toc149245877)

[4 Multihilo con Synchronized – Corredores 6](#_Toc149245878)

# Procesos

Implementa tres programas en Java:

1. Una aplicación llamada '**frecuencia**' que dado un texto recibido a través de su entrada estándar obtenga la frecuencia absoluta de cada una de las vocales 'a' 'e' 'o', tanto mayúsculas como minúsculas, es decir, la cantidad de veces que aparece cada una de ellas. En el caso de que una de las vocales no aparezca la frecuencia será 0. Por ejemplo: para la palabra “estupendo” el resultado debe ser  0 2 1 (0 indica el contador de la letra 'a', 2 indica el contador de la letra 'e', y 1 indica el contador de la letra 'o'). Otro ejemplo: para la palabra "Elefante" el resultado debe ser 1 3 0 como puedes observar el contador de 'e' ha tenido en cuenta tanto las mayúsculas como las minúsculas. El resto de vocales no se tienen en cuenta.
2. Una aplicación llamada '**generador**' que genere cadenas de texto formadas por caracteres del alfabeto mezclando mayúsculas y minúsculas. La cantidad de cadenas será indicada por el usuario al ejecutar la aplicación y su longitud será de 10 caracteres. Las cadenas se deben escribir en su salida estándar.
3. Una aplicación llamada '**lanzador**' que lance como máximo diez instancias de la aplicación '**generador**'. El usuario indicará el número de instancias (respetando el máximo establecido) y el número de cadenas a generar cada instancia.

# Fichero, escritura simultanea

Ten en cuenta que debes realizar sólo una aplicación por cada tipo, es decir, sólo una aplicación **frecuencia**, sólo un generador y una **lanzador**, que cumplan todas las partes de la tarea. **Se deberá entregar una carpeta separada con cada una de las aplicaciones frecuencia.jar, generador.jar y lanzador.jar.**

* Primera parte: implementa una aplicación que escriba en un fichero indicado por el usuario conjuntos de letras generadas de forma aleatoria (sin sentido real). Escribiendo cada conjunto de letras en una línea distinta. El número de conjuntos de letras a generar por el proceso, también será dado por el usuario en el momento de su ejecución. Esta aplicación se llamará "lenguaje" y como ejemplo, podrá ser invocada así:

java -jar lenguaje Nombre1 40 miFicheroDeLenguaje.txt

* + Primer parámetro: nombre dado a la instancia de lenguaje
  + Segundo parámetro: Conjunto de letras a generar
  + Cuando escribas la línea primero deberás escribir: “Escrito por X” siendo X el nombre pasado
* Segunda parte: implementa una aplicación, llamada 'colaborar', que lance al menos 10 instancias de la aplicación "lenguaje". Haciendo que todas ellas, colaboren en generar un gran fichero de palabras. Cada instancia generará un número creciente de palabras de 10, 20, 30, … Por supuesto, cada proceso seguirá escribiendo su palabra en una línea independiente de las otras. Es decir, si lanzamos 10 instancias de "lenguaje", al final, debemos tener en el fichero 10 + 20 + 30 + … + 100 = 550 líneas.

Utiliza objetos de tipo RandomAccessFile para el acceso a ficheros, recuerda que puede ser necesario que te posiciones al final del fichero para ir añadiendo valores en el fichero uno detrás de otro.

Tener en cuenta el bloqueo del fichero para escritura.

# Pajaros

## Enunciado

Tenemos pájaros de tres tipos: periquito, loro y gorrión.

Todos quieren cantar, pero tenemos diferentes niveles de educación en los pájaros:

1. Pájaros no educados, cantan cuando pueden, es decir, pueden cantar más de uno a la vez.
2. Los pájaros son educados y nunca cantan a la vez que los pájaros de su misma especie, pero sí a la vez que los pájaros de otra especie.
3. Los pájaros son más educados todavía y cantan por turnos, no cantando nunca a la vez, independientemente de la raza.
4. Un adiestrador muy bueno ha conseguido que los pájaros:
   1. Nunca cantan a la vez (independientemente de la raza del pájaro)
   2. Los pájaros se preparan para cantar, pero no empiezan hasta que su adiestrador les da el visto bueno (el adiestrador elige uno cualquiera para empezar). Como idea, el adiestrador puede esperar hasta que todos los pájaros estén preparados para cantar (ejemplo: espera 8 segundos antes de empezar).
   3. Cuando un pájaro de una especie termina de cantar da el turno a un pájaro cualquiera de otra especie (se elige aleatoriamente) siempre que haya uno de ellos esperando, si no hay ninguno de otra especie le da el turno a otro pájaro de su propia especie. Bonus si haces el código de forma que funcione con 3, 5 o 1000 razas de pajaros.
5. Estos pájaros no necesitan ni siquiera adiestrador. Empiezan a cantar en cuanto pueden sin que nadie les de ninguna señal, pero cumplen los mismos puntos que en el apartado anterior (recuerda que tardan 2 segundos en terminar de cantar)

Realiza uno o múltiples proyectos que implementen lo anterior.

* Antes de intentar cantar cada pájaro espera 5 segundos.
* El pájaro tarda 2 segundos en cantar (duerme el hilo cada vez que el pájaro cante) y debes escribir por pantalla que pájaro y de que raza ha cantado.
* Debes escribir cuantos segundos tarda cada método en terminar (recuerda esperar a que terminen todos los hilos para ello). Los segundos que dura cada tipo te servirá como indicativo de si es correcto o no el ejercicio.
* Eres libre de crear las clases que creas convenientes, pero al menos tiene que existir una clase “Pajaro” que extienda de Thread o Implemente Runnable y cada instancia de pájaro deberá lanzarse en un hilo diferente.

Ejemplo de salida, canto de cada pájaro y segundos totales. Hecho con 30 pájaros, unos 10 por cada raza.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

# Multihilo con Synchronized – Corredores

1. Crea un nuevo Proyecto. El paquete se llamará “tunombre.multitasking.carreranoexclusiva”. Dentro cread 3 clases:

* Clase Carrera que contendrá el programa principal.
* Clase Corredor heredará de **Thread**.
* Clase TrofeoGanador, con un único atributo **Boolean** estaElTrofeo, e inicializado a **true**.

2. Clase Carrera. Codificad en el main():

* Dos objetos instanciados con nombre “corredor1” y “corredor2”. Ambos objetos son instancias de la Clase Corredor.
* En la construcción del objeto se le pasa como parámetro un “**String** nombre”, el cuál será el nombre del corredor.
* Mostrad por pantalla (consola) un mensaje de que empieza la carrera.
* Ejecuta el método **run()** de ambos objetos (recuerda que el encargado de ejecutar el método **run()** de la clase Thread, o clases que hereden de **Thread**, es el método **start()** ) para que comience la ejecución multihilo.

3. Clase Corredor. Añade después de que corra (del bucle for) si el trofeo está disponible entonces pon el atributo esGanador a verdadero.

4. La Clase TrofeoGanador deberá implementar dos métodos sincronizados. Es hora de utilizar un mecanismo de sincronización sobre su atributo. De tal manera, que cuando un hilo esté accediendo a la información, ningún otro hilo pueda modificarla. Lo realizamos por medio de la palabra reservada **synchronized**. Y los dos métodos que contiene son:

● **Boolean** estaDisponible();

Este método devuelve si el atributo estaElTrofeo está aún a **true** en caso de que no se lo haya llevado ningún corredor. O contendrá el valor **false** si lo obtuvo otro corredor.

● **Boolean** obtenerTrofeo();

Este método devuelve true y marca el atributo de la clase a **false**. (Una vez que alguien se lleva el trofeo, éste ya no está como es lógico)

Un ejemplo del resultado de la ejecución:



¡Recuerda que el ganador o el orden de cómo avanzan los corredores cada 100m podría cambiar en cada ejecución!