DOCUMENTACIÓN TÉCNICA PROGRAMA CONTEO DE VOCALES

OMAR TORRES YUSTE 2º DAM

PROGRAMACIÓN PROCESOS Y SERVICIOS

Documentación técnica programa conteo de vocales

Índice

Descripción general

Este programa es una aplicación que utiliza el multiprocesamiento para analizar un conjunto de ficheros de texto, creando un proceso por cada fichero a procesar. Su objetivo principal es contar el número de vocales en todos los ficheros en la carpeta de ficheros.

App.java

En esta clase se inicia la aplicación y llama a la clase de padreProcesos que a su ves llama a hijoprocesos. Usando Process builder para ejecutar a PadreProcesos como un proceso, usamos pb.inheritIO() para ver la salida en consola del procesoPadre, usamos pb.start() para iniciarlo y p.waitfor() para esperar a que padreProcesos acabe su ejecución..

```java
public class App {

    /**
     * Inicio del programa llamada a PadreProcesos.
     * * Esta clase crea un proceso que ejecuta la clase {@link PadreProcesos},
     *
     * @author Omar
     * @version 1.0
     */
    Run | Debug
    public static void main(String[] args) {
        try {

            ProcessBuilder pb = new ProcessBuilder(
                    ...command:"java", "-cp", "bin;.", "PadreProcesos");

            pb.inheritIO(); // Ve la salida de hijos en consola

            Process p = pb.start();

            p.waitFor();

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

PadreProcesos.java

```java
public class PadreProcesos {

    Run | Debug
    public static void main(String[] args) {

        File carpeta = new File(pathname:"ficheros");
        File[] lista = carpeta.listFiles();
        ArrayList<File> ficherosValidos = new ArrayList<>();

        // No declaro ninguna de sus variables porque solo usare la funcion de borrar y
        // no necesito de ningun param
        Ficheros f = new Ficheros(ruta:null, id:null, lista:null);

        f.BorrarFicheros();

        if (lista == null) {
            System.out.println(x:"No existe la carpeta o está vacía.");
            return;
        }

        for (File fichero : lista) {
            String nombre = fichero.getName();
            if (nombre.matches(regex:"datos\\d+\\.txt")) {
                ficherosValidos.add(fichero);
            } else {
                System.out.println("Fichero con nombre incorrecto: " + nombre);
            }
        }

        // Si no hay ficheros válidos, salimos

        if (ficherosValidos.isEmpty()) {
            System.out.println(x:"No se encontraron ficheros válidos.");
            return;
        }

        int numeroFicheros = ficherosValidos.size();
```

```java
// Lanzar un hijo por cada fichero
for (int i = 0; i < numeroFicheros; i++) {
    String ruta = ficherosValidos.get(i).getAbsolutePath();
    String id = String.valueOf(i + 1);

    try {
        ProcessBuilder pb = new ProcessBuilder(
                ...command:"java", "-cp", "bin;.", "Hijo", ruta, id);

        pb.inheritIO();
        Process p = pb.start();
        p.waitFor();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// Contar numero de vocales

int totalGeneral = 0;

for (int i = 1; i <= numeroFicheros; i++) {
    File ficheroVocales = new File("vocales" + i + ".res");
    if (ficheroVocales.exists()) {
        try (BufferedReader br = new BufferedReader(new FileReader(ficheroVocales))) {
            String contenido = br.readLine();
            if (contenido != null && !contenido.trim().isEmpty()) {
                totalGeneral += Integer.parseInt(contenido.trim());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    } else {
        System.out.println("No se encontró el fichero: " + ficheroVocales.getName());
    }
}

// Guarda el total en un fichero

try (BufferedWriter bw = new BufferedWriter(new FileWriter(fileName:"totalVocales.res"))) {
    bw.write(String.valueOf(totalGeneral));
} catch (IOException e) {
    e.printStackTrace();
}
```

Esta clase gestiona la limpieza, la creación de subprocesos hijos y creación de txt con los números de vocales.

Gestionar el ciclo de vida completo del procesamiento de ficheros. Sus responsabilidades son:

1. Preparar el entorno eliminando ficheros de ejecuciones anteriores.
2. Validar y listar los ficheros de datos que deben ser procesados.
3. Delegar el trabajo, lanzando un proceso Hijo para cada fichero.
4. Recopilar y sumar los resultados devueltos por cada Hijo .
5. Generar el fichero final con el total de vocales.

Métodos principales

BorrarFicheros, limpia los ficheros residuales que puedan quedar de una ejecución anterior, se declara null porque aqui no voy a usar ni rutas, ni id ni lista.

```
Ficheros f = new Ficheros(ruta:null, id:null, lista:null);

f.BorrarFicheros();
```

VerificarFicheros, verifica que los ficheros que tengan su formato correcto se guarden en un nuevo arraylist donde estan los ficheros correctos, los que no tengan el nombre apropiado no se meten en el array y lanza un comentario en consola indicando que x fichero no cumple los requisitos.

```
if (lista == null) {
    System.out.println(x:"No existe la carpeta o está vacía.");
    return;
}

for (File fichero : lista) {
    String nombre = fichero.getName();
    if (nombre.matches(regex:"datos\\d+\\.txt")) {
        ficherosValidos.add(fichero);
    } else {
        System.out.println("Fichero con nombre incorrecto: " + nombre);
    }
}

// Si no hay ficheros válidos, salimos

if (ficherosValidos.isEmpty()) {
    System.out.println(x:"No se encontraron ficheros válidos.");
    return;
}
```

Lanzamiento de procesosHijos, crea un hijo por cada fichero y le asigna un id y le indica la ruta.

```java
// Lanzar un hijo por cada fichero
for (int i = 0; i < numeroFicheros; i++) {
    String ruta = ficherosValidos.get(i).getAbsolutePath();
    String id = String.valueOf(i + 1);

    try {
        ProcessBuilder pb = new ProcessBuilder(
                ...command:"java", "-cp", "bin;.", "Hijo", ruta, id);

        pb.inheritIO();
        Process p = pb.start();
        p.waitFor();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Recolección de Resultados, Cuando todos los subprocesos han terminado se crea otro bucle para leer los ficheros de los resultados y hacer la suma total y crear un fichero con la suma de todas las vocales **.**

```java
    // Contar numero de vocales

    int totalGeneral = 0;

    for (int i = 1; i <= numeroFicheros; i++) {
        File ficheroVocales = new File("vocales" + i + ".res");
        if (ficheroVocales.exists()) {
            try (BufferedReader br = new BufferedReader(new FileReader(ficheroVocales))) {
                String contenido = br.readLine();
                if (contenido != null && !contenido.trim().isEmpty()) {
                    totalGeneral += Integer.parseInt(contenido.trim());
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("No se encontró el fichero: " + ficheroVocales.getName());
        }
    }

    // Guarda el total en un fichero

    try (BufferedWriter bw = new BufferedWriter(new FileWriter(fileName:"totalVocales.res"))) {
        bw.write(String.valueOf(totalGeneral));
    } catch (IOException e) {
        e.printStackTrace();
    }

    System.out.println("Total de vocales: " + totalGeneral);
}
```

Hijo.java

Cada fichero se procesa utilizando un hijo, y esta clase se encarga de analizar su respectivo fichero y hacer los correspondientes metodos para tratar con ellos, llamando a clases como vocales o ficheros.

Main, arranca al hijo y se le pasa 2 argumentos de tipo string, 1 es el id y el otro es la ruta del fichero a tratar.

```
String rutaFichero = args[0];
String id = args[1];
```

f.LeerYGuardar(), lee el fichero datos y completa la lista palabras

```
f.LeerYGuardar();
```

f.GuardarEnFicheroMinuscula(), crea el fichero minusculaX.res con el contenido de datosX en minuscula

```
f.GuardarEnFicheroMinusculas();
```

v.contarVocalesTotales(palabras), llama el metodo de conteo de vocales y cuenta las vocales de la lista palabras y da como resultado un int que guardamos en totalvocales

```
int totalVocales = v.contarVocalesTotales(palabras);
```

inheritIO(), muestra los resultados de cada proceso en la terminal, mostrando cuantas palabras proceso y vocales.

```
System.out.println("Hijo " + id + " procesó " + palabras.size() + " palabras y " + totalVocales + " vocales ");
```

Ficheros.java

Aquí están todas las interacciones con los sistemas de archivos, tiene atributos como ruta, id y un arraylist, todo esto metido en el constructor para al llamarlo desde otra clase poder ir a tiro hecho.

BorrarFicheros(), es el sistema de borrado de ficheros residuales, lo cree en caso de tener 4 ficheros de datos y luego tener 3 por ejemplo, si no tengo esta función siempre quedaría 4 ficheros txt y haría mal el conteo por eso es importante esta función

```java
public void BorrarFicheros() {


    File carpeta = new File(pathname:"."); // o "." si quieres la carpeta actual
    File[] lista = carpeta.listFiles();

    if (lista != null) {
        for (File fichero : lista) {
            String nombre = fichero.getName().toLowerCase();
            if (nombre.endsWith(suffix:".res") || nombre.endsWith(suffix:".txt")) {
                boolean eliminado = fichero.delete();
                if (eliminado) {
                    System.out.println(" Borrado: " + fichero.getName());
                } else {
                    System.out.println(" No se pudo borrar: " + fichero.getName());
                }
            }
        }
    }
}
```

LeeryGuardar(), aquí lee palabra por palabra y lo añade al arraylist para tener guardada todas las palabras recortando los espacios ahí.

```java
public void LeerYGuardar() {

    try (BufferedReader br = new BufferedReader(new FileReader(ruta))) {
        String linea;
        while ((linea = br.readLine()) != null) {
            if (!linea.trim().isEmpty()) {
                lista.add(linea);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
        return;
    }
}
```

GuardarEnFicheroMinuscula() lee la lista y pasa las palabras a minúsculas para posteriormente pasarlo a un fichero llamado minusculaX.txt

```
public void GuardarEnFicheroMinusculas() {

    try (BufferedWriter bw = new BufferedWriter(new FileWriter("minusculas" + id + ".res"))) {
        for (String palabra : lista) {
            bw.write(palabra.toLowerCase() + "\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

guardarTotalVocales(int totalVocales) crea un fichero salido con el conteo de las vocales de cada fichero.

```
public void guardarTotalVocales(int totalVocales) {

    try (BufferedWriter bw = new BufferedWriter(new FileWriter("vocales" + id + ".res"))) {
        bw.write(String.valueOf(totalVocales));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Vocales.java

Esta clase se encarga de la utilidad para las vocales, aunque en mi caso solo tiene 1 función

contarVocalesTotales(ArrayList<String> lista), cuenta el número de vocales que ahí en el arraylist y compara si tiene las vocales que se le indica en su diccionario y se compara con IndexOf,, esto para también tener en cuenta tildes y otras anotaciones lingüísticas.

```java
public class Vocales {

    int numerovocales = 0;

    /**
     * Cuenta el total de vocales (normales y acentuadas) en una lista de palabras.
     *
     * @param lista Lista de palabras a analizar
     * @return Número total de vocales encontradas
     */
    public int contarVocalesTotales(ArrayList<String> lista) {
        int totalVocales = 0;
        String vocales = "aeiouáéíóúü";

        for (String palabra : lista) {
            for (int i = 0; i < palabra.length(); i++) {
                char c = palabra.toLowerCase().charAt(i);
                if (vocales.indexOf(c) != -1) {
                    totalVocales++;
                }
            }
        }

        return totalVocales;
    }
```