

Creación de procesos.

En Java es posible crear procesos utilizando algunas clases que el entorno ofrece para esta tarea. En este tema, veremos en profundidad la clase `ProcessBuilder`.

El ejemplo siguiente muestra como lanzar un proceso de Acrobat Reader:

```
public class LanzadorProcesos {
    public void ejecutar(String ruta){

        ProcessBuilder pb;
        try {
            pb = new ProcessBuilder(ruta);
            pb.start();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
    /**
     * @param args
     */
    public static void main(String[] args) {
        String ruta=
            "C:\\Program Files (x86)\\Adobe\\Reader
11.0\\Reader\\AcroRd32.exe";
        LanzadorProcesos lp=new LanzadorProcesos();
        lp.ejecutar(ruta);
        System.out.println("Finalizado");
    }

}
```

Supongamos que necesitamos crear un programa que aproveche al máximo el número de CPUs para realizar alguna tarea intensiva. Supongamos que dicha tarea consiste en sumar números.

Enunciado: crear una clase Java que sea capaz de sumar todos los números comprendidos entre dos valores incluyendo ambos valores.

Para resolverlo crearemos una clase `Sumador` que tenga un método que acepte dos números `n1` y `n2` y que devuelva la suma de todo el intervalo.

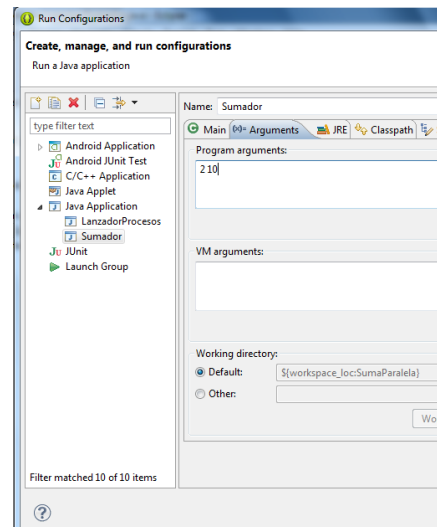
Además, incluiremos un método `main` que ejecute la operación de suma tomando los números de la línea de comandos (es decir, se pasan como argumentos al `main`).

El código de dicha clase podría ser algo así:

```
package com.ies;

public class Sumador {
    public int sumar(int n1, int n2){
        int resultado=0;
        for (int i=n1;i<=n2;i++){
            resultado=resultado+i;
        }
        return resultado;
    }
    public static void main(String[] args){
        Sumador s=new Sumador();
        int n1=Integer.parseInt(args[0]);
        int n2=Integer.parseInt(args[1]);
        int resultado=s.sumar(n1, n2);
        System.out.println(resultado);
    }
}
```

Para ejecutar este programa desde dentro de Eclipse es necesario indicar que deseamos enviar *argumentos* al programa. Por ejemplo, si deseamos sumar los números del 2 al 10, deberemos ir a la ventana «Run configuration» y en la pestaña «Arguments» indicar los argumentos (que en este caso son los dos números a indicar).



Modificando
los argumentos
del programa

Una vez hecha la prueba de la clase sumador, le quitamos el main, y crearemos una clase que sea capaz de lanzar varios procesos. La clase Sumador se quedará así:

```
public class Sumador {
    public int sumar(int n1, int n2){
        int resultado=0;
        for (int i=n1;i<=n2;i++){
            resultado=resultado+i;
        }
        return resultado;
    }
}
```

Y ahora tendremos una clase que lanza procesos de esta forma:

```
package com.ies;

public class Lanzador {
    public void lanzarSumador(Integer n1,
        Integer n2){
        String clase="com.ies.Sumador";
        ProcessBuilder pb;
        try {
```

```

        pb = new ProcessBuilder(
            "java", clase,
            n1.toString(),
            n2.toString());

        pb.start();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public static void main(String[] args){
    Lanzador l=new Lanzador();
    l.lanzarSumador(1, 51);
    l.lanzarSumador(51, 100);
    System.out.println("Ok");
}
}

```

Comunicación entre procesos.

Las operaciones multiproceso pueden implicar que sea necesario comunicar información entre muchos procesos, lo que obliga a la necesidad de utilizar mecanismos específicos de comunicación que ofrecerá Java o a diseñar alguno separado que evite los problemas que puedan aparecer.

En el ejemplo, el segundo proceso suele sobrescribir el resultado del primero, así que modificaremos el código del lanzador para que cada proceso use su propio fichero de resultados.

```

public class Lanzador {
    public void lanzarSumador(Integer n1,
        Integer n2, String fichResultado){
        String clase="com.ies.Sumador";
        ProcessBuilder pb;
        try {
            pb = new ProcessBuilder(
                "java", clase,
                n1.toString(),
                n2.toString());

```

```

        pb.redirectError(new
File("errores.txt"));
        pb.redirectOutput(new
File(fichResultado));
        pb.start();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
public static void main(String[] args){
    Lanzador l=new Lanzador();
    l.lanzarSumador(1, 5, "result1.txt");
    l.lanzarSumador(6,10, "result2.txt");
    System.out.println("Ok");
}
}

```

Cuando se lanza un programa desde Eclipse no ocurre lo mismo que cuando se lanza desde Windows. Eclipse trabaja con unos directorios predefinidos y puede ser necesario indicar a nuestro programa cual es la ruta donde hay que buscar algo.

Usando el método `.directory(new File("c:\\dir\\"))` se puede indicar a Java donde está el archivo que se desea ejecutar.

Ejercicio

Crear un programa que permita parametrizar el lanzamiento de sumadores, que vuelque el contenido de las sumas en ficheros y que permita al programa principal recuperar las sumas de los ficheros parciales.

En el listado siguiente se muestra la clase Sumador

```

package es.ies.multiproceso;

public class Sumador {
    /** Suma todos los valores incluidos
     * entre dos valores
     * @param n1 Limite 1

```

```

    * @param n2 Limite 2
    * @return La suma de dichos valores
    */
    public static int sumar(int n1, int n2){
        int suma=0;
        if (n1>n2){
            int aux=n1;
            n1=n2;
            n2=aux;
        }
        for (int i=n1; i<=n2; i++){
            suma=suma+i;
        }
        return suma;
    }

    public static void main(String[] args){
        int n1=Integer.parseInt(args[0]);
        int n2=Integer.parseInt(args[1]);
        int suma=sumar(n1, n2);
        System.out.println(suma);
        System.out.flush();
    }
}

```

En el listado siguiente se muestra la clase Main

```

public class Main {

    static final int NUM_PROCESOS=4;
    static final String PREFIJO_FICHEROS="fich";

    public static void lanzarSumador(
        int n1, int n2,String fichResultados)
    throws IOException{
        String comando;
        comando="es.ies.multiproceso.Sumador";

        File directorioSumador;
    }
}

```

```

        directorioSumador=new File("C:\\Users\\"+
        "ogomez\\workspace\\"+
        "MultiProceso1\\bin\\");
        File fichResultado=new File(fichResultados);
        ProcessBuilder pb;
        pb=new ProcessBuilder("java",
                                comando,
                                String.valueOf(n1),
                                String.valueOf(n2) );
        pb.directory(directorioSumador);
        pb.redirectOutput(fichResultado);
        pb.start();
    }

    public static int getResultadoFichero(
        String nombreFichero){

        int suma=0;
        try {
            FileInputStream fichero=
                new FileInputStream(

nombreFichero);

            InputStreamReader fir=
                new InputStreamReader(
                    fichero);

            BufferedReader br=new
BufferedReader(fir);

            String linea=br.readLine();
            suma= new Integer(linea);
            return suma;
        } catch (FileNotFoundException e) {
            System.out.println(
                "No se pudo abrir
"+nombreFichero);

        } catch (IOException e) {
            System.out.println(
                "No hay nada en
"+nombreFichero);
        }
    }

```

```

        return suma;
    }

    public static int getSumaTotal(int numFicheros){
        int sumaTotal=0;
        for (int i=1; i<=NUM_PROCESOS;i++){
            sumaTotal+=getResultadoFichero(
                PREFIJO_FICHEROS+String.valueOf(i) );
        }
        return sumaTotal;
    }

    /* Recibe dos parámetros y hará
    * la suma de los valores comprendidos
    * entre ambos parametros
    */
    public static void main(String[] args) throws
        IOException, InterruptedException{
        int n1=Integer.parseInt(args[0]);
        int n2=Integer.parseInt(args[1]);
        int salto=( n2 / NUM_PROCESOS ) - 1;
        for (int i=1;i<=NUM_PROCESOS; i++){
            System.out.println("n1:"+n1);
            int resultadoSumaConSalto=n1+salto;

            System.out.println("n2:"+resultadoSumaConSalto);
            lanzarSumador(n1, n1+salto ,

                PREFIJO_FICHEROS+String.valueOf(i));
            n1=n1 + salto + 1;
            System.out.println("Suma lanzada...");
        }
        Thread.sleep(5000);
        int sumaTotal=getSumaTotal(NUM_PROCESOS);
        System.out.println("La suma total es:"+
            sumaTotal);
    }
}

```


Ejercicio resuelto

Crear un programa que sea capaz de contar cuantas vocales hay en un fichero. El programa padre debe lanzar cinco procesos hijo, donde cada uno de ellos se ocupará de contar una vocal concreta (que puede ser minúscula o mayúscula). Cada subproceso que cuenta vocales deberá dejar el resultado en un fichero. El programa padre se ocupará de recuperar los resultados de los ficheros, sumar todos los subtotales y mostrar el resultado final en pantalla.

Operaciones con ficheros

Tanto en este ejercicio como en muchos otros vamos a necesitar realizar ciertas operaciones con ficheros. Para aumentar nuestra productividad utilizaremos una clase llamada `UtilidadesFicheros` que nos permita realizar ciertas tareas como las siguientes:

- Obtener un objeto `BufferedReader` a partir de un nombre de fichero de tipo `String`. Llamaremos a este método `getBufferedReader` y lo utilizaremos para poder manejar un fichero mediante una clase de muy alto nivel que nos facilita la lectura de ficheros. En el listado adjunto se puede consultar dicho método.

```
public static BufferedReader getBufferedReader(  
    String nombreFichero) throws  
FileNotFoundException {  
    FileReader lector;  
    lector = new FileReader(nombreFichero);  
    BufferedReader bufferedReader;  
    bufferedReader = new BufferedReader(lector);  
    return bufferedReader;  
}
```

- De la misma forma crearemos un método `getPrintWriter` que nos devuelva un objeto de la clase `PrintWriter` para poder realizar fácilmente operaciones de escritura en ficheros. En el listado adjunto se muestra el código.

```
public static PrintWriter getPrintWriter(  
    String nombreFichero) throws IOException {  
    PrintWriter printWriter;  
    FileWriter fileWriter;
```

```

        fileWriter = new FileWriter(nombreFichero);
        printWriter = new PrintWriter(fileWriter);
        return printWriter;
    //Fin de getPrintWriter
}

```

- Aunque nos estamos anticipando, puede ser útil tener un método que dado un nombre de fichero nos devuelva un `ArrayList<String>` con todas las líneas que hay en el fichero. **En este ejercicio no lo usaremos.** Este método podría implementarse así:

```

public static ArrayList<String> getLineasFichero(
    String nombreFichero) throws IOException {
    ArrayList<String> lineas = new ArrayList<String>();
    BufferedReader bfr = getBufferedReader(nombreFichero);
    //Leemos líneas del fichero...
    String linea = bfr.readLine();
    while (linea != null) {
        //Y las añadimos al array
        lineas.add(linea);
        linea = bfr.readLine();
    }
    //Fin del bucle que lee líneas
    return lineas;
}

```

Procesado de ficheros

Necesitaremos una clase `ProcesadorFichero` que nos permita procesar los ficheros de la forma pedida. Antes de crear un programa multiproceso empezaremos por crear una clase simple que nos resuelva este problema.

Dicha clase tendrá un método `hacerRecuento` que resuelva el problema de contar el número de apariciones en un fichero dejando el total de apariciones en un fichero de salida distintos. En el código adjunto podemos ver como podría implementarse dicho método.

```

public static ArrayList<String> getLineasFichero(
    String nombreFichero) throws IOException {

```

```

    ArrayList<String> lineas = new ArrayList<String>();
    BufferedReader bfr = getBufferedReader(nombreFichero);
    //Leemos líneas del fichero...
    String linea = bfr.readLine();
    while (linea != null) {
        //Y las añadimos al array
        lineas.add(linea);
        linea = bfr.readLine();
    }
    //Fin del bucle que lee líneas
    return lineas;
}

```

La clase `ProcesadorFichero` será lanzada desde otra clase que llamaremos `Lanzador`. Nuestra `ProcesadorFichero` recogerá en su método `main` los siguientes parámetros:

1. El nombre del fichero a procesar. Lo llamaremos `nombreFicheroEntrada` y estará en la posición 0 de los argumentos.
2. La letra de la que hay que hacer el recuento de apariciones. La llamaremos `letra` y estará en la posición 1.
3. El nombre del fichero donde se dejarán los resultados. Lo llamaremos `nombreFicheroResultado` y estará en la posición 2 de los argumentos.

El código del `main` se muestra a continuación.

```

/**
 * Dado un fichero pasado como argumento, contará cuantas
 * apariciones hay de una cierta vocal (pasada como
 * argumento)
 * y dejará la cantidad en otro fichero (también pasado como
 * argumento)
 * @throws IOException
 * @throws FileNotFoundException */
public static void main(String[] args) throws
FileNotFoundException, IOException {
    String nombreFicheroEntrada = args[0];
    String letra = args[1];
    String nombreFicheroResultado = args[2];
    hacerRecuento(nombreFicheroEntrada, letra,
nombreFicheroResultado);
}

```

```
//Fin del main
}
```

Lanzamiento de los procesos

La clase Lanzador tendrá un método main que se encargará de varias cosas:

1. Recoger el primer parámetro (`args[0]`), que contendrá el fichero a procesar.
2. Recoger el segundo parámetro, que contendrá el directorio de `CLASSPATH` donde habrá que buscar la clase `UtilidadesFicheros`.
3. Recoger el tercer parámetro, que contendrá el directorio donde habrá que buscar la clase `ProcesadorFicheros`.
4. Una vez recogidos los parámetros, se lanzarán los procesos utilizando la clase `ProcessBuilder`.
5. Los procesos se ejecutarán y después recogeremos los resultados de los ficheros.

En el código siguiente puede verse el método `main` de esta clase `Lanzador`. La recogida de resultados se deja como ejercicio:

```
public static void main(String[] args) throws
IOException, InterruptedException {
    String ficheroEntrada;
    ficheroEntrada = args[0];
    String classpathUtilidades;
    classpathUtilidades = args[1];
    String classpathProcesadorFichero;
    classpathProcesadorFichero = args[2];
    String[] vocales = { "A", "E", "I", "O", "U" };
    String classPath;
    classPath = classpathProcesadorFichero + ":" +
classpathUtilidades;
    System.out.println("Usando classpath:" + classPath);
    /* Se lanzan los procesos*/
    for (int i = 0; i < vocales.length; i++) {
        String fichErrores = "Errores_" + vocales[i] +
            ".txt";

        ProcessBuilder pb;
        pb = new ProcessBuilder("java", "-cp", classPath,
            "com.ies.ProcesadorFichero",
```

```

ficheroEntrada,
                                vocales[i], vocales[i] + ".txt");
    //Si hay algún error, almacenarlo en un fichero
    pb.redirectError(new File(fichErrores));
    pb.start();
    //Fin del for
}
/* Esperamos un poco*/
Thread.sleep(3000);
/* La recogida de resultados se deja como
   * ejercicio al lector. ;) */
}

```

Ejercicio propuesto (I)

Se desea crear un programa que procese ficheros aprovechando el paralelismo de la máquina. Se tienen cinco ficheros con los siguientes nombres:

- informatica.txt
- gerencia.txt
- contabilidad.txt
- comercio.txt
- rrhh.txt

En cada fichero hay una lista de cantidades enteras que representa las contabilidades de distintos departamentos. Hay una cantidad en cada línea. Se desea que el programa creado sume la cantidad total que suman todas las cantidades de los cinco ficheros haciendo uso del paralelismo.

Solución al ejercicio propuesto (I)

Este ejercicio es bastante parecido al anterior, con la salvedad de que ahora necesitaremos sumar cantidades en lugar de buscar elementos. Por lo demás, también generaremos un fichero de resultados por cada fichero de datos y los llamaremos igual pero añadiendo la extensión «.res». Es decir, la suma de las cantidades de `informatica.txt` se dejará en `informatica.txt.res`.

Una vez generados todos los fichero **Resultado** tendremos que sumar todas las cantidades de estos ficheros. Su estructura es muy simple, todos contienen una

única línea con una cantidad. Dado que esta operación de sumar cantidades almacenadas en distintos ficheros es una operación muy habitual, añadiremos un método a `UtilidadesFicheros` que se encargue de hacer esta operación. Obsérvese que usaremos tipos de datos `long` y no `int` dado que no sabemos como de grandes serán las cantidades a procesar.

A continuación se muestra el código de este método.

```
//Fin de getLineasFichero
public static long getSuma(String[]
                           listaNombresFichero) {
    long suma = 0;
    ArrayList<String> lineas;
    String lineaCantidad;
    long cantidad;
    for (String nombreFichero : listaNombresFichero) {
        try {
            //Recuperamos todas las lineas
            lineas = getLineasFichero(nombreFichero);
            //Pero solo nos interesa la primera
            lineaCantidad = lineas.get(0);
            //Convertimos la linea a número
            cantidad = Long.parseLong(lineaCantidad);
            //Y se incrementa la suma total
            suma = suma + cantidad;
        } catch (IOException e) {
            System.err.println("Fallo al procesar el fichero "
                               + nombreFichero);
            //Fin del catch
        }
        //Fin del for que recorre los nombres de fichero
    }
    return suma;
}
```

Una vez creado este método que usaremos al final, pasemos a crear una clase `ProcesadorContabilidad` que leerá un fichero, sumará las cantidades y las dejará en el fichero de resultados. Esta clase solo tendrá un método `main` que realizará esta tarea.

Como nuestro programa está formado por varias clases que están dispersas por distintos directorios, nuestro `main` tomará el `CLASSPATH` de los argumentos.

```
public static void main(String[] args) throws
IOException {
    String nombreFichero = args[0];
    String nombreFicheroResultado = args[1];
    ArrayList<String> cantidades;
    long total = 0;
    try {
        //Extraemos las cantidades
        cantidades = UtilidadesFicheros.getLineasFichero(
            nombreFichero);
        //Y las sumamos una por una
        for (String lineaCantidad : cantidades) {
            long cantidad = Long.parseLong(lineaCantidad);
            total = total + cantidad;
            //Fin del for que recorre las cantidades
        }
        //Almacenamos el total en un fichero
        PrintWriter pw;
        pw = UtilidadesFicheros.getPrintWriter(
            nombreFicheroResultado);
        pw.println(total);
        pw.close();
    } //Fin del try
    catch (IOException e) {
        System.err.println("No se pudo procesar el fichero "
            + nombreFichero);
        e.printStackTrace();
    }
    //Fin del main
}
```

Por último, solo queda crear la clase `Lanzador` que recogerá los nombres de fichero que hay que sumar y lanzará un proceso de la clase `ProcesadorContabilidad` para cada uno de ellos. Por último, sumará todos los resultados y los dejará en un fichero que llamaremos (por ejemplo) `Resultado_global.txt`

```

public static void main(String[] args) throws
IOException {
    String classpath = args[0];
    String[] ficheros = { "informatica.txt", "gerencia.txt",
"contabilidad.txt", "comercio.txt", "rrhh.txt" };
    //Los nombres de los ficheros de resultados
    //se generarán y luego se almacenarán aquí
    String[] ficherosResultado;
    ficherosResultado = new String[ficheros.length];
    /* Lanzamos los procesos*/
    ProcessBuilder[] constructores;
    constructores = new ProcessBuilder[ficheros.length];
    for (int i = 0; i < ficheros.length; i++) {
        String fichResultado, fichErrores;
        fichResultado = ficheros[i] + SUFIJO_RESULTADO;
        fichErrores = ficheros[i] + SUFIJO_ERRORES;
        ficherosResultado[i] = fichResultado;
        constructores[i] = new ProcessBuilder();
        constructores[i].command("java", "-cp", classpath,

"com.ies.ProcesadorContabilidad", ficheros[i],
                                fichResultado);
        //El fichero de errores se generará, aunque
        //puede que vacío
        constructores[i].redirectError(new File(
                                fichErrores));

        constructores[i].start();
        //Fin del for que recorre los ficheros
    }
    //Calculamos las sumas de cantidades
    long total = UtilidadesFicheros.getSuma(ficherosResultado);
    //Y las almacenamos
    PrintWriter pw =
UtilidadesFicheros.getPrintWriter(RESULTADOS_GLOBALES);
    pw.println(total);
    pw.close();
    //Fin del main
}

```


Gestión de procesos.

La gestión de procesos se realiza de dos formas **muy distintas** en función de los dos grandes sistemas operativos: Windows y Linux.

- En Windows toda la gestión de procesos se realiza desde el «Administrador de tareas» al cual se accede con Ctrl+Alt+Supr. Existen otros programas más sofisticados que proporcionan algo más de información sobre los procesos, como Process Explorer (antes conocido con el nombre de ProcessViewer).

Comandos para la gestión de procesos en sistemas libres y propietarios.

En sistemas Windows, no existen apenas comandos para gestionar procesos. Puede obligarse al sistema operativo a arrancar la aplicación asociada a un archivo con el comando **START**. Es decir, si se ejecuta lo siguiente:

```
START documento.pdf
```

se abrirá el visor de archivos PDF el cual cargará automáticamente el fichero `documento.pdf`

En GNU/Linux se puede utilizar un terminal de consola para la gestión de procesos, lo que implica que no solo se pueden arrancar procesos si no tambien detenerlos, reanudarlos, terminarlos y modificar su prioridad de ejecución.

- Para arrancar un proceso, simplemente tenemos que escribir el nombre del comando correspondiente. Desde GNU/Linux se pueden controlar los servicios que se ejecutan con un comando llamado **service**. Por ejemplo, se puede usar `sudo service apache2 stop` para parar el servidor web y `sudo service apache2 start` para volver a ponerlo en marcha. También se puede reiniciar un servicio (tal vez para que relea un fichero de configuración que hemos cambiado) con `sudo service apache2 restart`.
- Se puede detener y/o terminar un proceso con el comando **kill**. Se puede usar este comando para **terminar un proceso** sin guardar nada usando `kill -SIGKILL <numproceso>` o `kill -9 <numproceso>`. Se puede pausar un proceso con `kill -SIGSTOP <numproceso>` y reanuncarlo con `kill -SIGCONT`

- Se puede enviar un proceso a segundo plano con comandos como `bg` o al arrancar el proceso escribir el nombre del comando terminado en `&`.
- Se puede devolver un proceso a primer plano con el comando `fg`.

Prioridades

En sistemas como GNU/Linux se puede modificar la prioridad con que se ejecuta un proceso. Esto implica dos posibilidades

- Si pensamos que un programa que necesitamos ejecutar es muy importante podemos darle más prioridad para que reciba «más turnos» del planificador.
- Y por el contrario, si pensamos que un programa no es muy necesario podemos quitarle prioridad y reservar «más turnos de planificador» para otros posibles procesos.

El comando `nice` permite indicar prioridades entre -20 y 19. El -20 implica que un proceso reciba la **máxima prioridad**, y el 19 supone asignar la **mínima prioridad**

Sincronización entre procesos.

Cuando se lanza más de un proceso de una misma sección de código no se sabe qué proceso ejecutará qué instrucción en un cierto momento, lo que es muy peligroso:

```
int i,j;
i=0;
if (i>=2){
    i=i+1;
    j=j+1
}
System.out.println("Ok");
i=i*2;
j=j-1;
```

Si dos o más procesos avanzan por esta sección de código es perfectamente posible que unas veces nuestro programa multiproceso se ejecute bien y otras no.

En todo programa multiproceso pueden encontrarse estas zonas de código «peligrosas» que deben protegerse especialmente utilizando ciertos mecanismos.

El nombre global para todos los lenguajes es denominar a estos trozos «secciones críticas».

Mecanismos para controlar secciones críticas

Los mecanismos más típicos son los ofrecidos por UNIX/Windows:

- Semáforos.
- Colas de mensajes.
- Tuberías (pipes)
- Bloques de memoria compartida.

En realidad algunos de estos mecanismos se utilizan más para intercomunicar procesos, aunque para los programadores Java la forma de resolver el problema de la «sección crítica» es más simple.

En Java, si el programador piensa que un trozo de código es peligroso puede ponerle la palabra clave `synchronized` y la máquina virtual Java protege el código automáticamente.

```
/* La máquina virtual Java evitará que más de un proceso/hilo  
acceda a este método*/
```

```
synchronized  
    public void actualizarPension(int nuevoValor){  
        /*..trozo de código largo omitido*/  
        this.pension=nuevoValor  
    }
```

```
/* Otro ejemplo, ahora no hemos protegido un método entero,  
sino solo un pequeño trozo de código.*/
```

```
for (int i=0; i=i+1; i++){  
    /* Código omitido*/  
    synchronized {  
        i=i*2;  
        j=j+1;  
    }
```