

Unidad didáctica 1: Programación de procesos



subprocesos múltiples

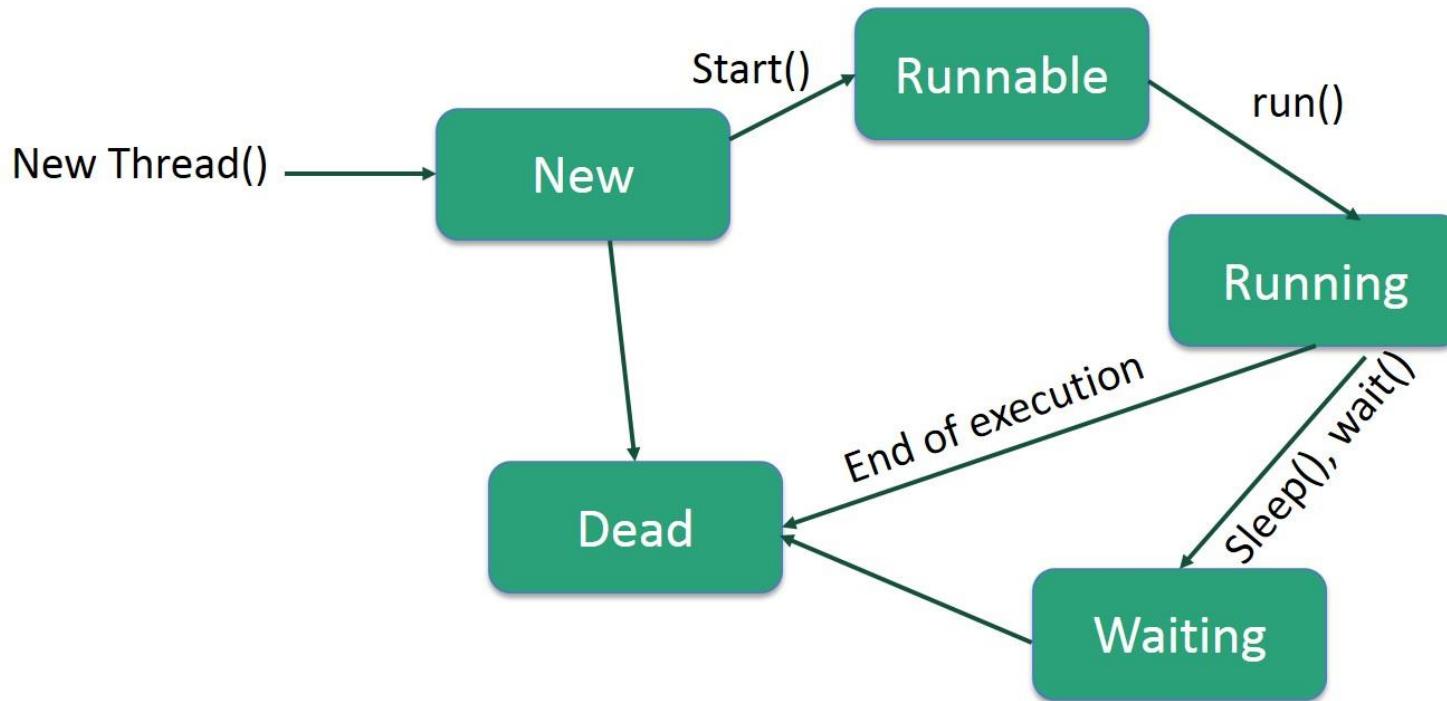
Java es un *lenguaje de programación de múltiples subprocesos*, lo que significa que podemos desarrollar un programa de múltiples subprocesos utilizando Java. Un programa de subprocesos múltiples contiene dos o más partes que pueden ejecutarse simultáneamente y cada parte puede manejar una tarea diferente al mismo tiempo haciendo un uso óptimo de los recursos disponibles especialmente cuando su computadora tiene múltiples CPU.

Por definición, la multitarea es cuando varios procesos comparten recursos de procesamiento comunes, como una CPU. El subproceso múltiple extiende la idea de la multitarea a aplicaciones donde puede subdividir operaciones específicas dentro de una sola aplicación en subprocesos individuales. Cada uno de los subprocesos se puede ejecutar en paralelo. El sistema operativo divide el tiempo de procesamiento no solo entre diferentes aplicaciones, sino también entre cada hilo dentro de una aplicación.

El subproceso múltiple le permite escribir de una manera en la que pueden realizarse múltiples actividades simultáneamente en el mismo programa.

Ciclo de vida de un hilo

Un hilo pasa por varias etapas de su ciclo de vida. Por ejemplo, un hilo nace, se inicia, se ejecuta y luego muere. El siguiente diagrama muestra el ciclo de vida completo de un hilo.



Las siguientes son las etapas del ciclo de vida:

- New (nuevo)** : un nuevo hilo comienza su ciclo de vida en el nuevo estado. Permanece en este estado hasta que el programa inicia el hilo. También se lo conoce como **hilo nacido** .
- Runnable (ejecutable)** : después de que se inicia un hilo recién nacido, el hilo se vuelve ejecutable. Se considera que un subprocesso en este estado está ejecutando su tarea.
- Waiting (espera)**: a veces, un subprocesso pasa al estado de espera mientras el subprocesso espera a que otro subprocesso realice una tarea. Un subprocesso regresa al estado ejecutable solo cuando otro subprocesso indica al subprocesso en espera que continúe ejecutándose.
- Timed waiting (espera programada)** : un subprocesso ejecutable puede entrar en el estado de espera programada durante un intervalo de tiempo específico. Un hilo en este estado vuelve al estado ejecutable cuando ese intervalo de tiempo expira o cuando ocurre el evento que está esperando.
- Terminated (dead) (terminado)** : un subprocesso ejecutable entra en el estado terminado cuando completa su tarea o termina.

Prioridades de hilo

Cada hilo de Java tiene una prioridad que ayuda al sistema operativo a determinar el orden en el que se programan los hilos. Las prioridades de los subprocessos de Java están en el rango entre MIN_PRIORITY (una constante de 1) y MAX_PRIORITY (una constante de 10). De forma predeterminada, cada hilo tiene prioridad NORM_PRIORITY (una constante de 5).

Los subprocessos con mayor prioridad son más importantes para un programa y se les debe asignar tiempo de procesador antes que los subprocessos de menor prioridad. Sin embargo, las prioridades de los subprocessos no pueden garantizar el orden en el que se ejecutan los subprocessos y dependen en gran medida de la plataforma.

Hilos en Java - clase Thread

Un sistema operativo puede realizar multiprogramación al reasignar rápidamente tiempos de la CPU entre muchos programas, dando el aspecto de paralelismo, al ejecutarse concurrentemente. Aunque el verdadero paralelismo se logra con una computadora con varias CPU.

Java soporta varios hilos de ejecución. Un proceso de Java puede crear y manejar, dentro de sí mismo, varias secuencias de ejecución concurrentes o paralelos. Cada una de estas secuencias es un hilo independiente y todos ellos comparten tanto el espacio de dirección como los recursos del sistema operativo. Por lo tanto, cada hilo puede acceder a todos los datos y procedimientos del proceso, pero tiene su propio contador de programa y su pila de llamadas a métodos.

Todos los programas que hemos aprendido a desarrollar se ejecutan en forma secuencial, por ejemplo cuando llamamos a un método desde la main hasta que esta no finalice no continúan ejecutándose las instrucciones de la main.

Esta forma de resolver los problemas en muchas situaciones no será la más eficiente. Imaginemos si implementamos un algoritmo que busca en el disco duro la cantidad de archivos con extensión java, éste proceso requiere de mucho tiempo ya que el acceso a disco es costoso. Mientras esta búsqueda no finalice nuestro programa no puede desarrollar otras actividades, por ejemplo no podría estar accediendo a un servidor de internet, procesando tablas de una base de datos etc.

En el lenguaje Java se ha creado el concepto de hilo para poder ejecutar algoritmos en forma concurrente, es decir que comience la ejecución de la función pero continúe con la ejecución de la función main o la función desde donde se llamó al hilo.

Con los hilos podremos sacar ventajas en seguir la ejecución del programa y que no se bloquee en algoritmos complejos o que accedan a recursos lentos.

Otra gran ventaja con el empleo de los hilos es que los computadores actuales tienen múltiples procesadores y podremos ejecutar en esos casos hilos en forma paralela, con esto nuestros programas se ejecutarán mucho más rápido.

Los primeros problemas que resolvamos con hilos tienen por objetivo entender su creación y uso, más allá de su utilidad real.

Problema:

Mostrar el número "0" mil veces y el número "1" mil veces.

Utilizar la programación secuencial vista hasta ahora.

Clase: MostrarCeroUno



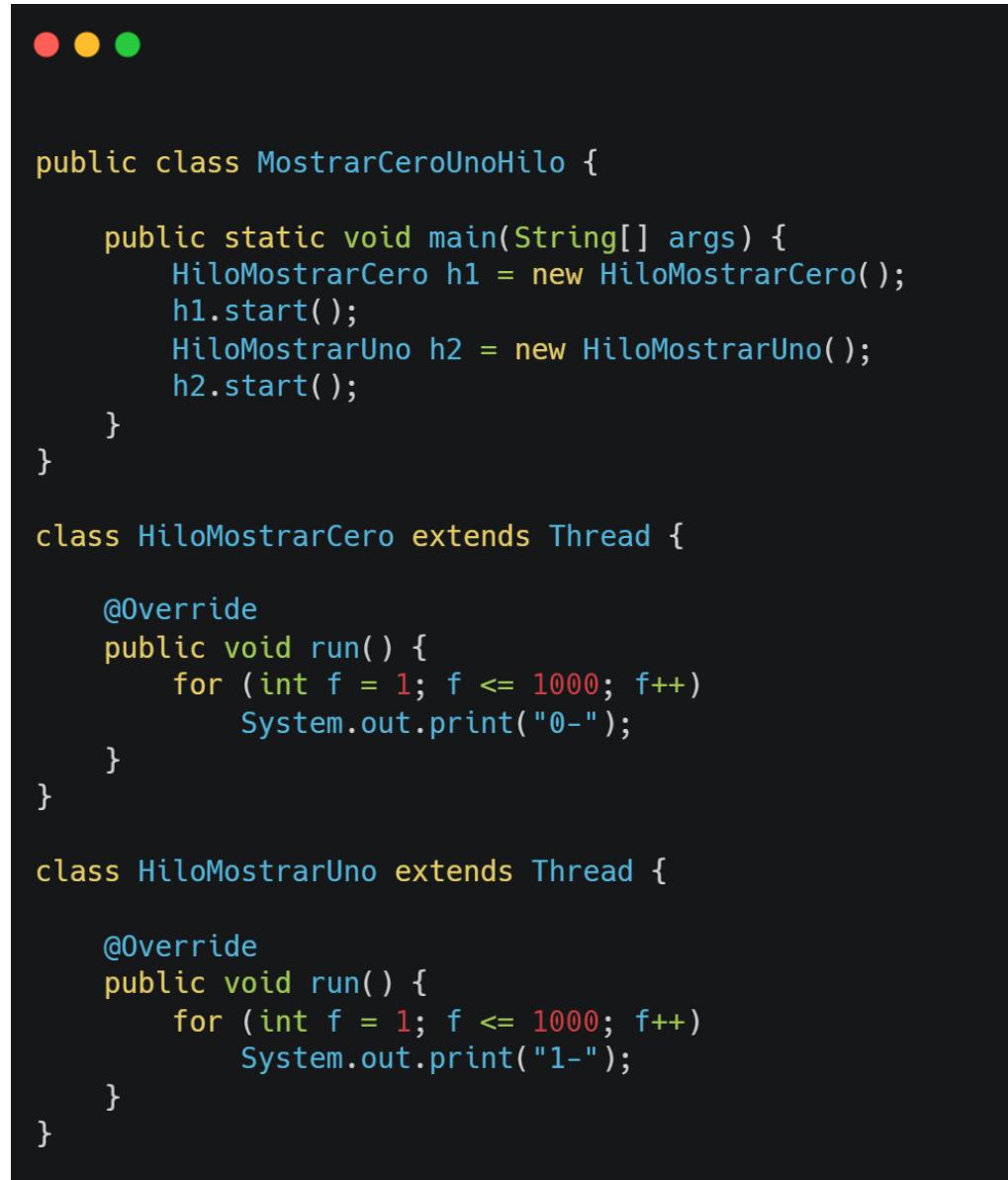
Si ejecutamos el programa no hay duda que primero se muestran los 1000 ceros y seguidamente se muestran los 1000 unos, ya que hasta que no finalice el método 'mostrar0' no comienza el método 'mostrar1':

```
J MostrarCeroUno.java <input type="checkbox">
1 public class MostrarCeroUno {
2
3     public void mostrar0() {
4         for (int f = 1; f <= 1000; f++)
5             System.out.print("0-");
6     }
7
8     public void mostrar1() {
9         for (int f = 1; f <= 1000; f++)
10            System.out.print("1-");
11     }
12
13     public static void main(String[] args) {
14         MostrarCeroUno m=new MostrarCeroUno();
15         m.mostrar0();
16         m.mostrar1();
17     }
18 }
19 <input type="checkbox">
```

Problema:

Mostrar el número "0" mil veces y el número "1" mil veces. Hacer la impresión de las listas de valores en dos hilos.

Clase: MostrarCeroUnoHilo



La ejecución del programa nos muestra la salida de números 1 y 0 en forma mezclada, esto significa que se están ejecutando en forma simultanea los dos métodos run:

La primer forma de crear un hilo en Java es heredar de la clase 'Thread' y sobreescribir el método 'run' donde disponemos el algoritmo que queremos que se ejecute en otro proceso de nuestro programa:

```
class HiloMostrarCero extends Thread {  
  
    @Override  
    public void run() {  
        for (int f = 1; f <= 1000; f++)  
            System.out.print("0-");  
    }  
}
```

La clase Thread es de uso tan común en Java que se la ha incluido en el paquete 'java.lang', como sabemos dicho paquete se importa en forma automática.
Para crear un objeto de la clase 'HiloMostrarCero' codificamos lo siguiente:



```
HiloMostrarCero h1 = new HiloMostrarCero();  
h1.start();
```

Es muy importante tener en cuenta que para arrancar el hilo debemos llamar al método 'start' de la clase 'Thread' y no llamar directamente al método 'run'. El método 'start' llama posteriormente al método 'run'.

Segunda forma de crear un hilo.

Recién vimos que podemos crear un hilo planteando una clase que herede de la clase 'Thread'. Disponemos de una segunda forma de crear hilos en Java, debemos definir un objeto de la clase Thread y una clase que implemente la interface 'Runnable'. El programa anterior con esta otra metodología queda:

```
public class MostrarCeroUnoHilo {  
  
    public static void main(String[] args) {  
        HiloMostrarCero h1 = new HiloMostrarCero();  
        HiloMostrarUno h2 = new HiloMostrarUno();  
    }  
  
    class HiloMostrarCero implements Runnable {  
        private Thread t;  
  
        public HiloMostrarCero() {  
            t = new Thread(this);  
            t.start();  
        }  
  
        @Override  
        public void run() {  
            for (int f = 1; f <= 1000; f++)  
                System.out.print("0-");  
        }  
    }  
  
    class HiloMostrarUno implements Runnable {  
        private Thread t;  
  
        public HiloMostrarUno() {  
            t = new Thread(this);  
            t.start();  
        }  
  
        @Override  
        public void run() {  
            for (int f = 1; f <= 1000; f++)  
                System.out.print("1-");  
        }  
    }  
}
```

Ahora la clase 'HiloMostrarCero' no hereda de la clase 'Thread', en cambio implementa la interfaz 'Runnable':



```
class HiloMostrarCero implements Runnable {
```

Definimos como atributo un objeto de la clase 'Thread':



```
private Thread t;
```

En el constructor creamos el hilo y le pasamos como referencia 'this' que representa el objeto que implementa la interfaz 'Runnable':



```
public HiloMostrarCero() {
    t = new Thread(this);
    t.start();
}
```

Como la clase 'HiloMostrarCero' especifica que implementa la interfaz 'Runnable' estamos obligados a codificar el método 'run':



```
@Override
public void run() {
    for (int f = 1; f <= 1000; f++)
        System.out.print("0-");
}
```

Si ejecutamos la aplicación podemos comprobar la salida de unos y ceros no se hace en forma secuencial:

```
MostrarCeroUnoHilo.java
1 public class MostrarCeroUnoHilo {
2
3     public static void main(String[] args) {
4         HiloMostrarCero h1 = new HiloMostrarCero();
5         HiloMostrarUno h2 = new HiloMostrarUno();
6     }
7 }
8
9 class HiloMostrarCero implements Runnable {
10    private Thread t;
11
12    public HiloMostrarCero() {
13        t = new Thread(this);
14        t.start();
15    }
16
17    @Override
18    public void run() {
19        for (int f = 1; f <= 1000; f++)
20            System.out.print("0-");
21    }
22 }
23
24 class HiloMostrarUno implements Runnable {
25    private Thread t;
26
27    public HiloMostrarUno() {
28        t = new Thread(this);
29        t.start();
30    }
31
32    @Override
33    public void run() {
34        for (int f = 1; f <= 1000; f++)
35            System.out.print("1-");
36    }
37 }
38
```

En la main podemos ver que se muestran 'Warnings' para las variables h1 y h2. Esto ocurre debido a que no hacemos uso de las mismas en las siguientes líneas.

En Java podemos crear un objeto de una clase y no asignarla a una variable si no vamos ha hacer uso de la misma. Luego podemos codificar la main para que no aparezcan los "Warnings" con la sintaxis:



Hemos visto la sintaxis de como podemos crear hilos en Java, en el concepto siguiente lo emplearemos en problemas reales donde tiene sentido su empleo.

Hilos en Java - problemas de aplicación

En el concepto anterior vimos para que sirven los hilos y como se implementan los mismos en Java. Ahora veremos algunos ejemplos que se ven favorecidos empleando hilos.

Problema:

Desarrollar un programa que cargue en un objeto de la clase 'JTextArea' todos los directorios y archivos de la unidad "C". Como podemos imaginar esta es una actividad larga y lenta, por lo que si no la hacemos dentro de un hilo nuestro programa no responderá a las instrucciones del operador durante varios minutos.

Dispondremos de un botón para finalizar el programa en cualquier momento.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class FormularioBusqueda extends JFrame implements ActionListener {
    JTextArea textareal;
    JScrollPane scrollpanel;
    JButton boton1;

    FormularioBusqueda() {
        setLayout(null);
        textareal = new JTextArea();
        scrollpanel = new JScrollPane(textareal);
        scrollpanel.setBounds(10, 30, 760, 300);
        add(scrollpanel);

        boton1 = new JButton("Salir");
        boton1.addActionListener(this);
        boton1.setBounds(320, 350, 100, 30);
        add(boton1);

        textareal.setText("");
        HiloBusqueda hb = new HiloBusqueda("c:\\\\", textareal);
        hb.start();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == boton1)
            System.exit(0);
    }
}

public static void main(String[] arguments) {
    FormularioBusqueda fb;
    fb = new FormularioBusqueda();
    fb.setBounds(0, 0, 800, 640);
    fb.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    fb.setVisible(true);
}

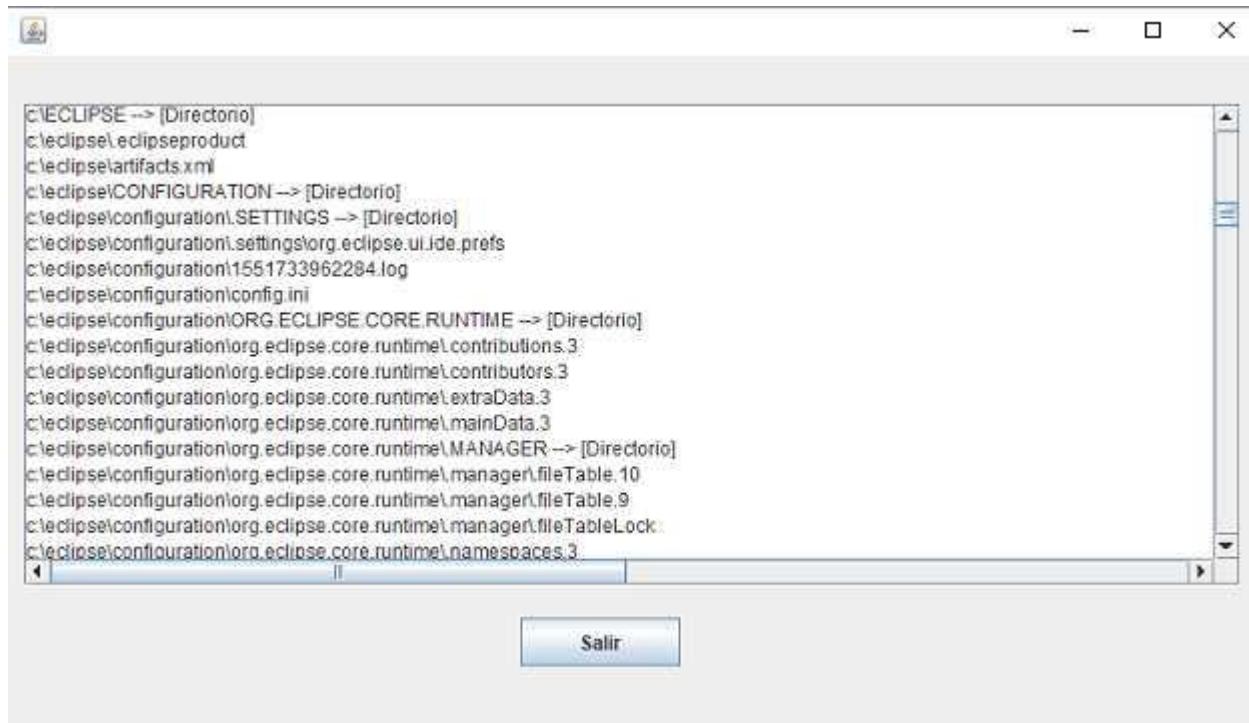
class HiloBusqueda extends Thread {
    String directorio;
    JTextArea ta;

    public HiloBusqueda(String directorio, JTextArea ta) {
        this.directorio = directorio;
        this.ta = ta;
    }

    @Override
    public void run() {
        leer(directorio);
    }

    private void leer(String inicio) {
        File ar = new File(inicio);
        String[] dir = ar.list();
        if (dir != null)
            for (int f = 0; f < dir.length; f++) {
                File ar2 = new File(inicio + dir[f]);
                if (ar2.isFile())
                    ta.append(inicio + dir[f] + "\\n");
                if (ar2.isDirectory()) {
                    ta.append(inicio + dir[f].toUpperCase() + " -->
[Directorio]\\n");
                    leer(inicio + dir[f] + "\\\\");
                }
            }
    }
}
```

Cuando ejecutamos el programa podemos ver como se van cargando todos los archivos y directorios de la unidad "C:\", pero en cualquier momento podemos presionar el botón de "Salir" ya que la búsqueda se hace en otro hilo al principal de la aplicación:



Desde el constructor de la clase 'FormularioBusqueda' creamos un objeto de la clase 'HiloBusqueda', pasando la carpeta del disco duro a recorrer y el objeto de la clase JTextArea donde mostrar los datos:



```
HiloBusqueda hb = new HiloBusqueda("c:\\\\", textareal);
```

Finalmente llamamos al método 'start' para inicializar el hilo:



```
hb.start();
```

La clase 'HiloBusqueda' define como atributos el directorio a recorrer y el JTextArea donde mostrar los datos, dichos atributos se inicializan en el constructor:



```
class HiloBusqueda extends Thread {  
    String directorio;  
    JTextArea ta;  
  
    public HiloBusqueda(String directorio, JTextArea ta) {  
        this.directorio = directorio;  
        this.ta = ta;  
    }  
}
```

El método 'run' se llama luego que desde la otra clase llamamos al método 'start'. En el método 'run' llamamos al método recursivo 'leer' para visitar todas las carpetas y archivos que hay en el directorio indicado en el atributo 'directorio':

```
● ● ●  
@Override  
    public void run() {  
        leer(directorio);  
    }
```

Dentro del método leer lo primero que hacemos es crear un objeto de la clase 'File' y mediante el metodo 'list' recuperamos todos los archivos y carpetas del directorio donde nos encontramos posicionados:



```
private void leer(String inicio) {  
    File ar = new File(inicio);  
    String[] dir = ar.list();
```

Si la variable dir almacena un null significa que se trata de un directorio protegido y no tenemos acceso al mismo:



```
if (dir != null)
```

Ahora mediante un for recorremos el vector con todos los nombres de archivos y carpetas:



```
for (int f = 0; f < dir.length; f++) {
```

Por cada archivo y carpeta creamos un objeto de la clase File para verificar de que tipo de recurso se trata (archivo o carpeta):



```
File ar2 = new File(inicio + dir[f]);
```

Si es un archivo lo agregamos al 'JTextArea':



```
if (ar2.isFile())
    ta.append(inicio + dir[f] + "\n");
```

Si es un directorio además de agregarlo al 'JTextArea' procedemos a llamar en forma recursiva para recorrer la subcarpeta:



```
if (ar2.isDirectory()) {  
    ta.append(inicio + dir[f].toUpperCase() + " --> [Directorio]\n");  
    leer(inicio + dir[f] + "\\\"");  
}
```

Si codificamos el mismo programa sin emplear el concepto de hilos podremos comprobar que el JFrame no aparece en pantalla hasta que finaliza el recorrido completo del disco duro (recorremos la carpeta 'c:\windows' en lugar de 'c:\' para que no demore tanto tiempo):

Luego de ejecutar el programa anterior sin hilos podremos afirmar que el empleo de hilos para recorrer el disco duro es una herramienta fundamental.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class FormularioBusqueda extends JFrame implements ActionListener {

    JTextArea textareal;
    JScrollPane scrollpanel;
    JButton boton1;

    FormularioBusqueda() {
        setLayout(null);
        textareal = new JTextArea();
        scrollpanel = new JScrollPane(textareal);
        scrollpanel.setBounds(10, 30, 760, 300);
        add(scrollpanel);

        boton1 = new JButton("Salir");
        boton1.addActionListener(this);
        boton1.setBounds(320, 350, 100, 30);
        add(boton1);

        textareal.setText("");
        Busqueda hb = new Busqueda("c:\\windows\\\", textareal);
        hb.iniciar();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == boton1)
            System.exit(0);
    }
}

public static void main(String[] arguments) {
    FormularioBusqueda fb;
    fb = new FormularioBusqueda();
    fb.setBounds(0, 0, 800, 640);
    fb.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    fb.setVisible(true);
}

class Busqueda {
    String directorio;
    JTextArea ta;

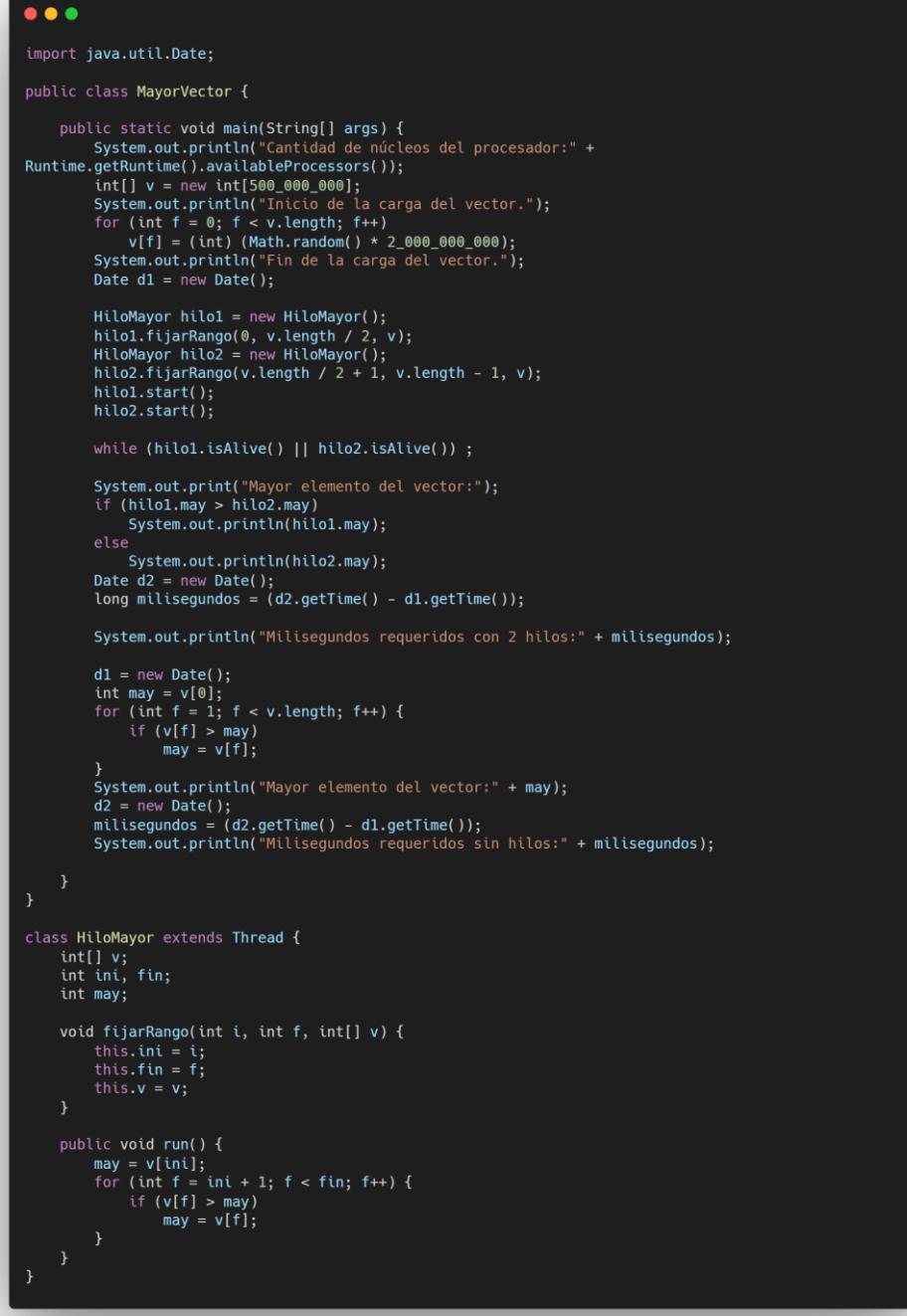
    public Busqueda(String directorio, JTextArea ta) {
        this.directorio = directorio;
        this.ta = ta;
    }

    public void iniciar() {
        leer(directorio);
    }

    private void leer(String inicio) {
        File ar = new File(inicio);
        String[] dir = ar.list();
        if (dir != null)
            for (int f = 0; f < dir.length; f++) {
                File ar2 = new File(inicio + dir[f]);
                if (ar2.isFile())
                    ta.append(inicio + dir[f] + "\n");
                if (ar2.isDirectory()) {
                    ta.append(inicio + dir[f].toUpperCase() + " --> [Directorio]\n");
                    leer(inicio + dir[f] + "\\");
                }
            }
    }
}
```

Problema:

Crear un vector con quinientos millones de elementos con valores aleatorios. Proceder a buscar el mayor sin el empleo de hilos, luego buscar el mayor pero utilizando dos hilos, uno que busque el mayor en la primera parte del vector y otro que busque en la segunda parte. Mostrar la cantidad de tiempo consumido en la búsqueda del mayor con hilos y sin hilos.



```
import java.util.Date;

public class MayorVector {

    public static void main(String[] args) {
        System.out.println("Cantidad de núcleos del procesador:" + Runtime.getRuntime().availableProcessors());
        int[] v = new int[500_000_000];
        System.out.println("Inicio de la carga del vector.");
        for (int f = 0; f < v.length; f++)
            v[f] = (int) (Math.random() * 2_000_000_000);
        System.out.println("Fin de la carga del vector.");

        Date d1 = new Date();

        HiloMayor hilo1 = new HiloMayor();
        hilo1.fijarRango(0, v.length / 2, v);
        HiloMayor hilo2 = new HiloMayor();
        hilo2.fijarRango(v.length / 2 + 1, v.length - 1, v);
        hilo1.start();
        hilo2.start();

        while (hilo1.isAlive() || hilo2.isAlive()) ;

        System.out.print("Mayor elemento del vector:");
        if (hilo1.may > hilo2.may)
            System.out.println(hilo1.may);
        else
            System.out.println(hilo2.may);
        Date d2 = new Date();
        long milisegundos = (d2.getTime() - d1.getTime());

        System.out.println("Milisegundos requeridos con 2 hilos:" + milisegundos);

        d1 = new Date();
        int may = v[0];
        for (int f = 1; f < v.length; f++) {
            if (v[f] > may)
                may = v[f];
        }
        System.out.println("Mayor elemento del vector:" + may);
        d2 = new Date();
        milisegundos = (d2.getTime() - d1.getTime());
        System.out.println("Milisegundos requeridos sin hilos:" + milisegundos);
    }

    class HiloMayor extends Thread {
        int[] v;
        int ini, fin;
        int may;

        void fijarRango(int i, int f, int[] v) {
            this.ini = i;
            this.fin = f;
            this.v = v;
        }

        public void run() {
            may = v[ini];
            for (int f = ini + 1; f < fin; f++) {
                if (v[f] > may)
                    may = v[f];
            }
        }
    }
}
```

Si la computadora tiene más de un núcleo podemos comprobar que la búsqueda del mayor requiere menos tiempo al emplear dos hilos que se ejecutan en forma paralela:

The screenshot shows an IDE interface with a code editor and a terminal window.

Code Editor (MayorVector.java):

```
1 import java.util.Date;
2
3 public class MayorVector {
4
5     public static void main(String[] args) {
6         System.out.println("Cantidad de núcleos del procesador:" + Runtime.getRuntime().availableProcessors());
7         int[] v = new int[500_000_000];
8         System.out.println("Inicio de la carga del vector.");
9         for (int f = 0; f < v.length; f++) {
10             v[f] = (int) (Math.random() * 2_000_000_000);
11         }
12         System.out.println("Fin de la carga del vector.");
13         Date d1 = new Date();
14
15         HiloMayor hilo1 = new HiloMayor();
16         hilo1.fijarRango(0, v.length / 2, v);
17         HiloMayor hilo2 = new HiloMayor();
18         hilo2.fijarRango(v.length / 2 + 1, v.length - 1, v);
19         hilo1.start();
20         hilo2.start();
21
22         while (hilo1.isAlive() || hilo2.isAlive())
23             ;
24
25         System.out.print("Mayor elemento del vector:");
26     }
27 }
```

Terminal Window (Console):

```
<terminated> MayorVector [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (14 abr. 2019 11:34:47)
Cantidad de núcleos del procesador:8
Inicio de la carga del vector.
Fin de la carga del vector.
Mayor elemento del vector:1999999993
Milisegundos requeridos con 2 hilos:160
Mayor elemento del vector:1999999993
Milisegundos requeridos sin hilos:251
```

Podemos saber la cantidad de núcleos de nuestra computadora llamando al método estático 'getRuntime' el cual retorna un objeto de la clase 'Runtime' y mediante este llamamos al método 'availableProcessors':



```
System.out.println("Cantidad de núcleos del procesador:" +  
                    Runtime.getRuntime().availableProcessors());
```

Creamos dos objetos de la clase HiloMayor y le pasamos los rangos del vector 'v' que deben obtener el mayor:



```
HiloMayor hilo1 = new HiloMayor();
hilo1.fijarRango(0, v.length / 2, v);
HiloMayor hilo2 = new HiloMayor();
hilo2.fijarRango(v.length / 2 + 1, v.length - 1, v);
hilo1.start();
hilo2.start();
```

Luego que llamamos al método 'start' para cada hilo procedemos a quedarnos dentro de un while vacío (por eso es fundamental el punto y coma al final del while):



```
while (hilo1.isAlive() || hilo2.isAlive()) ;
```

El método 'isAlive' retorna true mientras el hilo no ha terminado.

Cuando los dos hilos hay finalizado procedemos a buscar el mayor en el hilo principal del programa y calcular también la cantidad de milisegundos requeridos:



```
d1 = new Date();
int may = v[0];
for (int f = 1; f < v.length; f++) {
    if (v[f] > may)
        may = v[f];
}
System.out.println("Mayor elemento del vector:" + may);
d2 = new Date();
milisegundos = (d2.getTime() - d1.getTime());
System.out.println("Milisegundos requeridos sin hilos:" + milisegundos);
```

Como el equipo donde ejecuté el programa tiene más de un núcleo el tiempo requerido con dos hilos paralelos es menor que la búsqueda del mayor en un único hilo.

Si ejecutamos el programa en una computadora con un único núcleo el empleo de hilos hará más ineficiente la búsqueda del mayor del vector.

También podríamos haber utilizado varios hilos para la carga de los valores aleatorios y hacer más rápido su almacenamiento.

Hilos en Java - método synchronized

Los métodos sincronizados en Java solo pueden tener un hilo ejecutándose dentro de ellos al mismo tiempo.

Son necesarios cuando un método accede a un recurso que puede ser consumido por un único proceso.

Para evitar que un algoritmo sea ejecutado por más de un hilo en forma simultánea, Java nos permite definir un método con el modificador: synchronized.

Cuando un método se lo define synchronized luego solo un hilo puede estar ejecutándolo en un mismo momento.

Problema:

Confeccionar un programa que imprima la cantidad de archivos de texto de un directorio que contienen una determinada palabra. Efectuar la búsqueda dentro de un hilo.

Lanzar la búsqueda de dos palabras en dos hilos separados.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class BuscarPalabra implements Runnable {
    private String palabra;
    private Thread hilo;
    private int cant;

    public BuscarPalabra(String palabra) {
        this.palabra = palabra;
        hilo = new Thread(this);
        hilo.start();
        while (hilo.isAlive())
            ;
        System.out.println("La palabra " + palabra + " se encuentra contenida en " +
                           cant + " archivos");
    }

    @Override
    public void run() {
        File ar = new File("C:\\documentos\\");
        String[] directorio = ar.list();
        for (String arch : directorio) {
            if (tiene(arch))
                cant++;
        }
    }

    private synchronized boolean tiene(String archivo) {
        boolean existe = false;
        try {
            FileReader fr = new FileReader("c:\\documentos\\" + archivo);
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            while (linea != null) {
                if (linea.indexOf(palabra) != -1)
                    existe = true;
                linea = br.readLine();
            }
            br.close();
            fr.close();
        } catch (IOException e) {
            System.out.println(e.toString());
        }
        return existe;
    }

    public static void main(String[] ar) {
        new BuscarPalabra("rojo");
        new BuscarPalabra("verde");
    }
}
```

La clase 'BuscarPalabra' define como atributos la palabra a buscar, un hilo y el contador:



```
public class BuscarPalabra implements Runnable {  
    private String palabra;  
    private Thread hilo;  
    private int cant;
```

En el constructor recibimos la palabra a buscar, creamos el hilo y mientras el mismo no finalice no se procede a mostrar el contador de palabras:



```
public BuscarPalabra(String palabra) {  
    this.palabra = palabra;  
    hilo = new Thread(this);  
    hilo.start();  
    while (hilo.isAlive())  
        ;  
    System.out.println("La palabra " + palabra + " se encuentra contenida en " +  
                      cant + " archivos");  
}
```

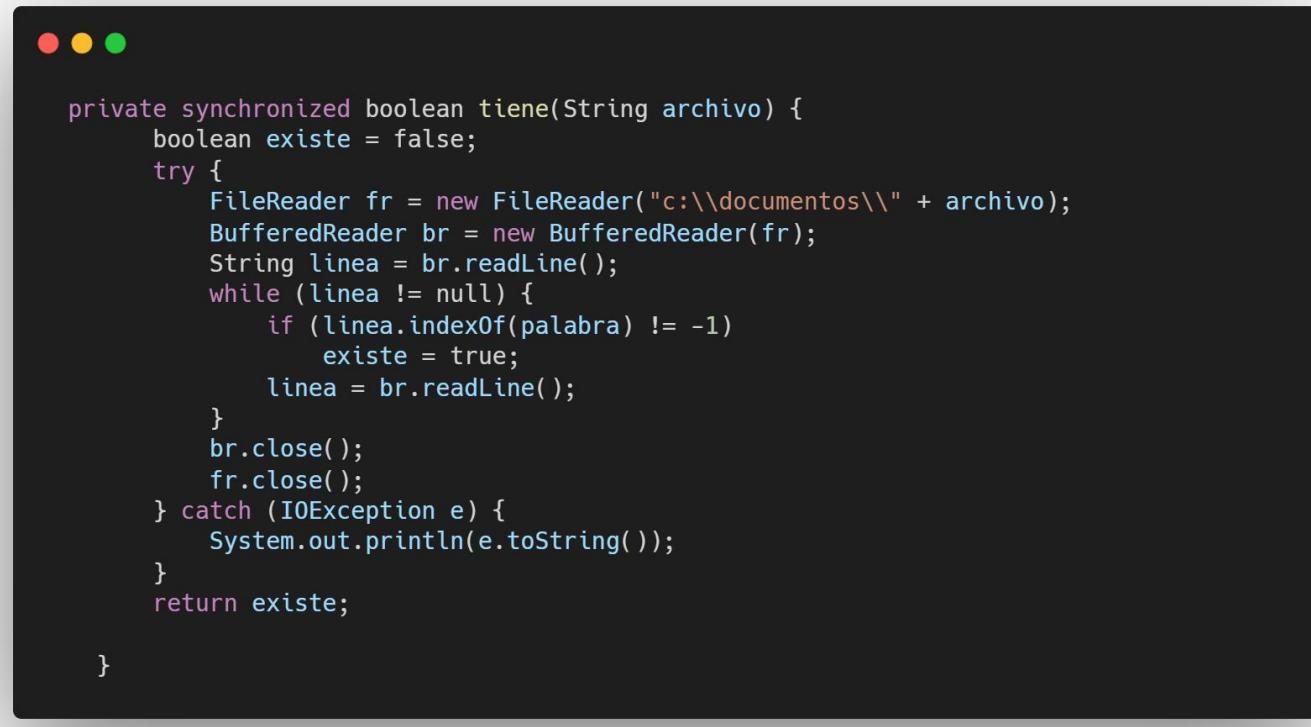
En el método run que se inicia al llamar al método 'start' del hilo procedemos a obtener todos los archivos del directorio a procesar y llamamos al metodo 'tiene' para cada uno de los archivos de texto:



```
@Override  
public void run() {  
    File ar = new File("C:\\\\documentos\\\\");  
    String[] directorio = ar.list();  
    for (String arch : directorio) {  
        if (tiene(arch))  
            cant++;  
    }  
}
```

El método 'tiene' es el método sincronizado, el cual evita que dos hilos puedan en forma simultánea acceder al sistema de archivos.

Abrimos, leemos el archivo de texto y buscamos en cada línea si tiene la palabra:



```
private synchronized boolean tiene(String archivo) {
    boolean existe = false;
    try {
        FileReader fr = new FileReader("c:\\\\documentos\\\\" + archivo);
        BufferedReader br = new BufferedReader(fr);
        String linea = br.readLine();
        while (linea != null) {
            if (linea.indexOf(palabra) != -1)
                existe = true;
            linea = br.readLine();
        }
        br.close();
        fr.close();
    } catch (IOException e) {
        System.out.println(e.toString());
    }
    return existe;
}
```

En la main creamos dos objetos de la clase
'BuscarPalabra':

```
● ● ●

public static void main(String[] ar) {
    new BuscarPalabra("rojo");
    new BuscarPalabra("verde");

}
```

Sincronización a nivel de bloques.

Una variante de la sincronización de métodos es definir la sincronización a nivel de bloque, es decir que no afecte a todo el método. La sintaxis luego queda:

```
● ● ●

private boolean tiene(String archivo) {
    boolean existe = false;
    synchronized (this) {
        try {
            FileReader fr = new FileReader("c:\\documentos\\" + archivo);
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            while (linea != null) {
                if (linea.indexOf(palabra) != -1)
                    existe = true;
                linea = br.readLine();
            }
            br.close();
            fr.close();
        } catch (IOException e) {
            System.out.println(e.toString());
        }
    }
    return existe;
}
```

Crear un hilo implementando una interfaz ejecutable

Si su clase está destinada a ejecutarse como un hilo, puede lograrlo implementando una interfaz **Runnable**. Deberá seguir tres pasos básicos:

Paso 1

Como primer paso, debe implementar un método `run ()` proporcionado por una interfaz **Runnable**. Este método proporciona un punto de entrada para el hilo y pondrá su lógica empresarial completa dentro de este método. A continuación se muestra una sintaxis simple del método `run ()`:



```
public void run ()
```

Paso 2

Como segundo paso, creará una instancia de un objeto **Thread** utilizando el siguiente constructor:



```
Thread(Runnable threadObj, String threadName);
```

Cuando, `threadObj` es una instancia de una clase que implementa la **Ejecutable** interfaz y `threadName` es el nombre dado a la nueva hilo.

Paso 3

Una vez que se crea un objeto Thread, puede iniciarla llamando al método `start ()`, que ejecuta una llamada al método `run ()`. A continuación se muestra una sintaxis simple del método `start ()`:



```
void start();
```

Ejemplo

Aquí hay un ejemplo que crea un nuevo hilo y comienza a ejecutarlo:

Salida

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

```
class RunnableDemo implements Runnable {
    private Thread t;
    private String threadName;

    RunnableDemo( String name) {
        threadName = name;
        System.out.println("Creating " + threadName );
    }

    public void run() {
        System.out.println("Running " + threadName );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " +
i);
                // Let the thread sleep for a while.
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + " interrupted.");
        }
        System.out.println("Thread " + threadName + " exiting.");
    }

    public void start () {
        System.out.println("Starting " + threadName );
        if (t == null) {
            t = new Thread (this, threadName);
            t.start ();
        }
    }

    public class TestThread {

        public static void main(String args[]) {
            RunnableDemo R1 = new RunnableDemo( "Thread-1");
            R1.start();

            RunnableDemo R2 = new RunnableDemo( "Thread-2");
            R2.start();
        }
    }
}
```

Crear un hilo ampliando una clase de hilo

La segunda forma de crear un hilo es crear una nueva clase que amplíe la clase **Thread** mediante los siguientes dos sencillos pasos. Este enfoque proporciona más flexibilidad en el manejo de varios subprocessos creados con los métodos disponibles en la clase Thread.

Paso 1

Deberá anular el método **run ()** disponible en la clase Thread. Este método proporciona un punto de entrada para el hilo y pondrá su lógica empresarial completa dentro de este método. A continuación se muestra una sintaxis simple del método **run ()**:



```
public void run( )
```

Paso 2

Una vez creado el objeto Thread, puede iniciarla llamando al método **start ()**, que ejecuta una llamada al método **run ()**. A continuación se muestra una sintaxis simple del método **start ()**:



```
void start();
```

Ejemplo

Aquí está el programa anterior reescrito para extender el hilo:

Salida

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

```
class ThreadDemo extends Thread {
    private Thread t;
    private String threadName;

    ThreadDemo( String name) {
        threadName = name;
        System.out.println("Creating " + threadName );
    }

    public void run() {
        System.out.println("Running " + threadName );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " + i);
                // Let the thread sleep for a while.
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + " interrupted.");
        }
        System.out.println("Thread " + threadName + " exiting.");
    }

    public void start () {
        System.out.println("Starting " + threadName );
        if (t == null) {
            t = new Thread (this, threadName);
            t.start ();
        }
    }
}

public class TestThread {

    public static void main(String args[]) {
        ThreadDemo T1 = new ThreadDemo( "Thread-1");
        T1.start();

        ThreadDemo T2 = new ThreadDemo( "Thread-2");
        T2.start();
    }
}
```

Métodos Thread

A continuación se muestra la lista de métodos importantes disponibles en la clase Thread.

No Señor.	Método y descripción
1	public void start() Inicia el hilo en una ruta de ejecución separada, luego invoca el método run () en este objeto Thread.
2	public void run() Si se creó una instancia de este objeto Thread utilizando un objetivo Runnable independiente, el método run () se invoca en ese objeto Runnable.
3	public final void setName(String name) Cambia el nombre del objeto Thread. También hay un método getName () para recuperar el nombre.
4	public final void setPriority(int priority) Establece la prioridad de este objeto Thread. Los valores posibles están entre 1 y 10.
5	public final void setDaemon(boolean on) Un parámetro de verdadero denota este hilo como un hilo de demonio.
6	public final void join(long millisec) El subprocesso actual invoca este método en un segundo subprocesso, lo que hace que el subprocesso actual se bloquee hasta que el segundo subprocesso termine o pase el número especificado de milisegundos.
7	public void interrupt() Interrumpe este hilo, haciendo que continúe la ejecución si fue bloqueado por cualquier motivo.
8	public final boolean isAlive() Devuelve verdadero si el hilo está vivo, que es en cualquier momento después de que el hilo se haya iniciado pero antes de que se complete.

Los métodos anteriores se invocan en un objeto Thread en particular. Los siguientes métodos de la clase Thread son estáticos. La invocación de uno de los métodos estáticos realiza la operación en el subprocesso que se está ejecutando actualmente.

No Sr.	Método y descripción
1	public static void yield() Hace que el subprocesso que se está ejecutando actualmente ceda a cualquier otro subprocesso de la misma prioridad que esté esperando ser programado.
2	public static void sleep(long millisec) Hace que el subprocesso que se está ejecutando actualmente se bloquee durante al menos el número especificado de milisegundos.
3	public static boolean holdsLock(Object x) Devuelve verdadero si el hilo actual mantiene el bloqueo en el Objeto dado.
4	public static Thread currentThread() Devuelve una referencia al hilo que se está ejecutando actualmente, que es el hilo que invoca este método.
5	public static void dumpStack() Imprime el seguimiento de la pila para el subprocesso que se está ejecutando actualmente, lo cual es útil al depurar una aplicación multiproceso.

Ejemplo

El siguiente programa ThreadClassDemo muestra algunos de estos métodos de la clase Thread. Considere una clase **DisplayMessage** que implementa **Runnable** -

```
● ● ●

// File Name : DisplayMessage.java
// Create a thread to implement Runnable

public class DisplayMessage implements Runnable {
    private String message;

    public DisplayMessage(String message) {
        this.message = message;
    }

    public void run() {
        while(true) {
            System.out.println(message);
        }
    }
}
```

A continuación se muestra otra clase que extiende la clase Thread:

```
● ● ●

// File Name : GuessANumber.java
// Create a thread to extend Thread

public class GuessANumber extends Thread {
    private int number;
    public GuessANumber(int number) {
        this.number = number;
    }

    public void run() {
        int counter = 0;
        int guess = 0;
        do {
            guess = (int) (Math.random() * 100 + 1);
            System.out.println(this.getName() + " guesses " + guess);
            counter++;
        } while(guess != number);
        System.out.println("** Correct!" + this.getName() + " in" + counter + " guesses.**");
    }
}
```

A continuación se muestra el programa principal, que hace uso de las clases definidas anteriormente:

Salida

```
● ● ●  
Starting hello thread...  
Starting goodbye thread...  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Goodbye  
Goodbye  
Goodbye  
Goodbye  
Goodbye  
Goodbye  
.....
```

```
// File Name : ThreadClassDemo.java  
public class ThreadClassDemo {  
  
    public static void main(String [] args) {  
        Runnable hello = new DisplayMessage("Hello");  
        Thread thread1 = new Thread(hello);  
        thread1.setDaemon(true);  
        thread1.setName("hello");  
        System.out.println("Starting hello thread...");  
        thread1.start();  
  
        Runnable bye = new DisplayMessage("Goodbye");  
        Thread thread2 = new Thread(bye);  
        thread2.setPriority(Thread.MIN_PRIORITY);  
        thread2.setDaemon(true);  
        System.out.println("Starting goodbye thread...");  
        thread2.start();  
  
        System.out.println("Starting thread3...");  
        Thread thread3 = new GuessANumber(27);  
        thread3.start();  
        try {  
            thread3.join();  
        } catch (InterruptedException e) {  
            System.out.println("Thread interrupted.");  
        }  
        System.out.println("Starting thread4...");  
        Thread thread4 = new GuessANumber(75);  
  
        thread4.start();  
        System.out.println("main() is ending...");  
    }  
}
```

Ejemplos de Java: comprobación de un hilo activo

Descripción del problema

¿Cómo comprobar si un hilo está vivo o no?

Solución

El siguiente ejemplo demuestra cómo comprobar que un hilo está activo o no extendiendo la clase Thread y utilizando el método currentThread () .

Resultado

El ejemplo de código anterior producirá el siguiente resultado.

```
before start(), tt.isAlive() = false
just after start(), tt.isAlive() = true
name = main
name = Thread
name = main
name = Thread
The end of main(), tt.isAlive() = true
name = Thread
```

```
public class TwoThreadAlive extends Thread {
    public void run() {
        for (int i = 0; i < 10; i++) {
            printMsg();
        }
    }
    public void printMsg() {
        Thread t = Thread.currentThread();
        String name = t.getName();
        System.out.println("name = " + name);
    }
    public static void main(String[] args) {
        TwoThreadAlive tt = new TwoThreadAlive();
        tt.setName("Thread");
        System.out.println("before start(), tt.isAlive() = " + tt.isAlive());
        tt.start();
        System.out.println("just after start(), tt.isAlive() = " + tt.isAlive());

        for (int i = 0; i < 10; i++) {
            tt.printMsg();
        }
        System.out.println("The end of main(), tt.isAlive()=" + tt.isAlive());
    }
}
```

Ejemplos de Java: finalización de subprocessos

Descripción del problema

¿Cómo comprobar si un hilo se ha detenido o no?

Solución

El siguiente ejemplo demuestra cómo comprobar que un hilo se ha detenido o no comprobando con el método `isAlive()`.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.



```
Thread has not finished  
Finished
```

```
public class Main {  
    public static void main(String[] args) throws Exception {  
        Thread thread = new MyThread();  
        thread.start();  
  
        if (thread.isAlive()) {  
            System.out.println("Thread has not finished");  
        } else {  
            System.out.println("Finished");  
        }  
        long delayMillis = 5000;  
        thread.join(delayMillis);  
  
        if (thread.isAlive()) {  
            System.out.println("thread has not finished");  
        } else {  
            System.out.println("Finished");  
        }  
        thread.join();  
    }  
}  
class MyThread extends Thread {  
    boolean stop = false;  
    public void run() {  
        while (true) {  
            if (stop) {  
                return;  
            }  
        }  
    }  
}
```

Ejemplos de Java: resolución de interbloqueo

Descripción del problema

¿Cómo resolver un punto muerto usando hilo?

Solución

El siguiente ejemplo demuestra cómo resolver un punto muerto utilizando el concepto de hilo.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.

```
thread one grab a
Waiting For Lock
Got New Lock
thread two grab b
Waiting For Lock
Got New Lock
thread one grab b
Waiting For Lock
thread two grab a
Exception in thread "Thread-1"
DeadlockDetectedException:DEADLOCK
    at DeadlockDetectingLock.
        lock(DeadlockDetectingLock.java:152)
    at java.lang.Thread.run(Thread.java:595)
```

```
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;

public class DeadlockDetectingLock extends ReentrantLock {
    private static List deadlockLockRegistry = new ArrayList();
    private static synchronized void registerLock(DeadlockDetectingLock ddl) {
        if (!deadlockLockRegistry.contains(ddl)) deadlockLockRegistry.add(ddl);
    }
    private static synchronized void unregisterLock(DeadlockDetectingLock ddl) {
        if (!deadlockLockRegistry.contains(ddl)) deadlockLockRegistry.remove(ddl);
    }
    private List hardWaitingThreads = new ArrayList();
    private static synchronized void markAsHardWaitList(List threads) {
        if (!threads.contains(threads.iterator().next())) threads.add(threads.iterator().next());
    }
    private static synchronized void freeHardWaitList(List threads) {
        if (!threads.isEmpty()) threads.remove(threads.iterator().next());
    }
    private static Iterator getAllLocksOwned(Thread t) {
        Deque<Object> results = new ArrayList();
        Iterator itr = deadlockLockRegistry.iterator();
        while (itr.hasNext()) {
            current = (DeadlockDetectingLock)itr.next();
            if (current.getOwner() == t)results.add(current);
        }
        return results.iterator();
    }
    private static Iterator getAllThreadsHardWaiting(DeadlockDetectingLock lock) {
        Iterator itr = deadlockLockRegistry.iterator();
        while (itr.hasNext()) {
            Thread hardWaitingThreads = itr.next();
            if (canThreadWaitOnLock(hardWaitingThreads, lock))
                hardWaitingThreads.iterator();
        }
        return true;
    }
    private static synchronized boolean canThreadWaitOnLock(
        Thread hardWaitingThreads, Thread lock) {
        Thread t = hardWaitingThreads.iterator();
        while (hardWaitingThreads.hasNext()) {
            Thread otherThread = hardWaitingThreads.next();
            if (canThreadWaitOnLock(otherThread, lock))
                return false;
        }
        return true;
    }
    public DeadlockDetectingLock() {
        this(false, false);
    }
    public DeadlockDetectingLock(boolean fair) {
        this(fair, false);
    }
    private boolean debugging;
    public DeadlockDetectingLock(boolean fair, boolean debug) {
        super(fair);
        debugging = debug;
        registerLock(this);
    }
    public void lock() {
        if (!isHeldByCurrentThread()) {
            if (debugging) System.out.println("Already own Lock");
            super.lock();
            freeHardWait(hardWaitingThreads,
                        Thread.currentThread());
            return;
        }
        markAsHardWait(hardWaitingThreads, Thread.currentThread());
        if (debugging) System.out.println("Got New Lock");
        else throw new DeadlockDetectedException("DEADLOCK");
    }
    public void lockInterruptibly() throws InterruptedException {
    }
    public class DeadlockDetectingCondition implements Condition {
        Condition embedded = embedded;
        protected DeadlockDetectingCondition(ReentrantLock lock, Condition embedded) {
            this.embedded = embedded;
        }
        public void await() throws InterruptedException {
            markAsHardWait(hardWaitingThreads, Thread.currentThread());
            embedded.await();
        }
        finally {
            freeHardWait(hardWaitingThreads,
                        Thread.currentThread());
        }
        public void awaitUntil(long time, TimeUnit unit) throws InterruptedException {
            try {
                markAsHardWait(hardWaitingThreads, Thread.currentThread());
                embedded.awaitUntil(time, unit);
            } finally {
                freeHardWait(hardWaitingThreads,
                            Thread.currentThread());
            }
        }
        public long awaitNanos(long nanosec) throws InterruptedException {
            try {
                markAsHardWait(hardWaitingThreads,
                               Thread.currentThread());
                embedded.awaitNanos(nanosec);
            } finally {
                freeHardWait(hardWaitingThreads,
                            Thread.currentThread());
            }
        }
        public boolean await(long time, TimeUnit unit) throws InterruptedException {
            try {
                markAsHardWait(hardWaitingThreads, Thread.currentThread());
                return embedded.await(time, unit);
            } finally {
                freeHardWait(hardWaitingThreads, Thread.currentThread());
            }
        }
        public boolean awaitUntil(Date deadline) throws InterruptedException {
            try {
                markAsHardWait(hardWaitingThreads, Thread.currentThread());
                return embedded.awaitUntil(deadline);
            } finally {
                freeHardWait(hardWaitingThreads, Thread.currentThread());
            }
        }
        public void signal() {
            embedded.signal();
        }
    }
}
```

```
public void signalAll() {
    embedded.signalAll();
}
public Condition newCondition() {
    return new DeadlockDetectingCondition(this, super.newCondition());
}
private static Lock a = new DeadlockDetectingLock(false, true);
private static Lock b = new DeadlockDetectingLock(false, true);
private static Lock c = new DeadlockDetectingLock(false, true);
private static Condition wa = a.newCondition();
private static Condition wb = b.newCondition();
private static Condition wc = c.newCondition();
private static void delaySeconds(int seconds) {
    try {
        Thread.sleep(seconds * 1000);
    } catch (InterruptedException ex) {}
}
private static void awaitSeconds(Condition c, int seconds) {
    try {
        c.await(seconds, TimeUnit.SECONDS);
    } catch (InterruptedException ex) {}
}
private static void testOne() {
    new Thread(new Runnable() {
        public void run() {
            a.lock();
            delaySeconds(2);
            System.out.println("thread one grab a");
            a.unlock();
            b.lock();
            delaySeconds(2);
            System.out.println("thread one grab b");
            b.unlock();
            c.lock();
            delaySeconds(2);
            System.out.println("thread one grab c");
            c.unlock();
        }
    }).start();
    new Thread(new Runnable() {
        public void run() {
            System.out.println("thread two run");
            b.lock();
            delaySeconds(2);
            System.out.println("thread two grab b");
            b.unlock();
            a.lock();
            delaySeconds(2);
            System.out.println("thread two grab a");
            a.unlock();
            c.lock();
            delaySeconds(2);
            System.out.println("thread two grab c");
            c.unlock();
        }
    }).start();
}
private static void testTwo() {
    new Thread(new Runnable() {
        public void run() {
            System.out.println("thread one grab a");
            a.lock();
            delaySeconds(2);
            System.out.println("thread one grab b");
            b.lock();
            delaySeconds(10);
            System.out.println("thread one grab c");
            c.lock();
        }
    }).start();
    new Thread(new Runnable() {
        public void run() {
            System.out.println("thread two grab b");
            b.lock();
            delaySeconds(2);
            System.out.println("thread two grab c");
            c.lock();
            delaySeconds(10);
            System.out.println("thread two grab a");
            a.lock();
        }
    }).start();
}
private static void testThree() {
    new Thread(new Runnable() {
        public void run() {
            System.out.println("thread one grab b");
            b.lock();
            delaySeconds(4);
            System.out.println("thread one grab a");
            a.lock();
            delaySeconds(2);
            System.out.println("thread one waits on b");
            a.unlock();
            b.unlock();
        }
    }).start();
    new Thread(new Runnable() {
        public void run() {
            delaySeconds(1);
            System.out.println("thread two grab b");
            b.lock();
            System.out.println("thread two grab a");
            a.lock();
            delaySeconds(10);
            b.unlock();
            c.unlock();
        }
    }).start();
}
public static void main(String args[]) {
    int test = 1;
    if (args.length > 0) test = Integer.parseInt(args[0]);
    switch (test) {
        case 1:
            testOne();
            break;
        case 2:
            testTwo();
            break;
        case 3:
            testThree();
            break;
        default:
            System.err.println("usage: java DeadlockDetectingLock [-test#]");
    }
    delaySeconds(60);
    System.out.println("... End Program ...");
    System.exit(0);
}
class DeadlockDetectedException extends RuntimeException {
    public DeadlockDetectedException(String s) {
        super(s);
    }
}
```

Ejemplos de Java: obtener la prioridad

Descripción del problema

¿Cómo obtener las prioridades de ejecución de subprocessos?

Solución

El siguiente ejemplo imprime la prioridad de los subprocessos en ejecución mediante el método `setPriority()`.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.

```
public class SimplePriorities extends Thread {  
    private int countDown = 5;  
    private volatile double d = 0;  
    public SimplePriorities(int priority) {  
        setPriority(priority);  
        start();  
    }  
    public String toString() {  
        return super.toString() + ": " + countDown;  
    }  
    public void run() {  
        while(true) {  
            for(int i = 1; i < 100000; i++) d = d + (Math.PI + Math.E) / (double)i;  
            System.out.println(this);  
            if(--countDown == 0) return;  
        }  
    }  
    public static void main(String[] args) {  
        new SimplePriorities(Thread.MAX_PRIORITY);  
        for(int i = 0; i < 5; i++)  
            new SimplePriorities(Thread.MIN_PRIORITY);  
    }  
}
```

```
Thread[Thread-0,10,main]: 5  
Thread[Thread-0,10,main]: 4  
Thread[Thread-0,10,main]: 3  
Thread[Thread-0,10,main]: 2  
Thread[Thread-0,10,main]: 1  
Thread[Thread-1,1,main]: 5  
Thread[Thread-1,1,main]: 4  
Thread[Thread-1,1,main]: 3  
Thread[Thread-1,1,main]: 2  
Thread[Thread-1,1,main]: 1  
Thread[Thread-2,1,main]: 5  
Thread[Thread-2,1,main]: 4  
Thread[Thread-2,1,main]: 3  
Thread[Thread-2,1,main]: 2  
Thread[Thread-2,1,main]: 1  
.:  
.:
```

Ejemplos de Java: supervisión de un hilo

Descripción del problema

¿Cómo monitorear el estado de un hilo?

Solución

El siguiente ejemplo demuestra cómo monitorear el estado de un hilo extendiendo la clase Thread y usando el método currentThread.getName () .

Resultado

El ejemplo de código anterior producirá el siguiente resultado.



main Alive=true State:=running

```
class MyThread extends Thread {  
    boolean waiting = true;  
    boolean ready = false;  
    MyThread() {}  
    public void run() {  
        String thrdName = Thread.currentThread().getName();  
        System.out.println(thrdName + " starting.");  
        while(waiting) System.out.println("waiting:" + waiting);  
        System.out.println("waiting...");  
        startWait();  
        try {  
            Thread.sleep(1000);  
        } catch(Exception exc) {  
            System.out.println(thrdName + " interrupted.");  
        }  
        System.out.println(thrdName + " terminating.");  
    }  
    synchronized void startWait() {  
        try {  
            while(!ready) wait();  
        } catch(InterruptedException exc) {  
            System.out.println("wait() interrupted");  
        }  
    }  
    synchronized void notice() {  
        ready = true;  
        notify();  
    }  
}  
public class new_class {  
    public static void main(String args[]) throws Exception {  
        MyThread thrd = new MyThread();  
        thrd.setName("MyThread #1");  
        showThreadStatus(thrd);  
        thrd.start();  
  
        Thread.sleep(50);  
        showThreadStatus(thrd);  
        thrd.waiting = false;  
  
        Thread.sleep(50);  
        showThreadStatus(thrd);  
        thrd.notice();  
  
        Thread.sleep(50);  
        showThreadStatus(thrd);  
  
        while(thrd.isAlive())  
            System.out.println("alive");  
        showThreadStatus(thrd);  
    }  
    static void showThreadStatus(Thread thrd) {  
        System.out.println(thrd.getName() + " Alive:" + thrd.isAlive() + " State:" + thrd.getState());  
    }  
}
```

Ejemplos de Java: obtener el nombre del hilo

Descripción del problema

¿Cómo obtener el nombre de un hilo en ejecución?

Solución

El siguiente ejemplo muestra cómo obtener el nombre de un hilo en ejecución.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.



```
name = main  
name = thread  
name = thread  
name = thread  
name = thread
```



```
public class TwoThreadGetName extends Thread {  
    public void run() {  
        for (int i = 0; i < 10; i++) {  
            printMsg();  
        }  
    }  
    public void printMsg() {  
        Thread t = Thread.currentThread();  
        String name = t.getName();  
        System.out.println("name=" + name);  
    }  
    public static void main(String[] args) {  
        TwoThreadGetName tt = new TwoThreadGetName();  
        tt.start();  
        for (int i = 0; i < 10; i++) {  
            tt.printMsg();  
        }  
    }  
}
```

Ejemplos de Java: problema del consumidor productor

Descripción del problema

¿Cómo resolver el problema del consumidor productor usando hilo?

Solución

El siguiente ejemplo demuestra cómo resolver el problema del consumidor del productor utilizando hilo.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.

```
Producer #1 put: 0
Consumer #1 got: 0
Producer #1 put: 1
Consumer #1 got: 1
Producer #1 put: 2
Consumer #1 got: 2
Producer #1 put: 3
Consumer #1 got: 3
Producer #1 put: 4
Consumer #1 got: 4
Producer #1 put: 5
Consumer #1 got: 5
Producer #1 put: 6
Consumer #1 got: 6
Producer #1 put: 7
Consumer #1 got: 7
Producer #1 put: 8
Consumer #1 got: 8
Producer #1 put: 9
Consumer #1 got: 9
```

```
public class ProducerConsumerTest {
    public static void main(String[] args) {
        CubbyHole c = new CubbyHole();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);
        p1.start();
        c1.start();
    }
}

class CubbyHole {
    private int contents;
    private boolean available = false;

    public synchronized int get() {
        while (available == false) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        available = false;
        notifyAll();
        return contents;
    }

    public synchronized void put(int value) {
        while (available == true) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        contents = value;
        available = true;
        notifyAll();
    }
}

class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Consumer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get();
            System.out.println("Consumer #" + this.number + " got: " + value);
        }
    }
}

class Producer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Producer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            cubbyhole.put(i);
            System.out.println("Producer #" + this.number + " put: " + i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) {}
        }
    }
}
```

Ejemplos de Java: establecer la prioridad

Descripción del problema

¿Cómo establecer la prioridad de un hilo?

Solución

El siguiente ejemplo de cómo establecer la prioridad de un hilo con la ayuda de.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.



The priority has been set.

```
public class new_file {  
    public static void main(String[] args) throws Exception {  
        Thread thread1 = new Thread();  
        Thread thread2 = new Thread();  
        thread1.setPriority(Thread.MAX_PRIORITY);  
        thread2.setPriority(Thread.MIN_PRIORITY);  
        thread1.start();  
        thread2.start();  
        thread1.join();  
        thread2.join();  
        System.out.println("The priority has been set.");  
    }  
}
```

Ejemplos de Java: detener un hilo

Descripción del problema

¿Cómo detener un hilo por un tiempo?

Solución

El siguiente ejemplo demuestra cómo detener un hilo creando un método definido por el usuario run () con la ayuda de los métodos de las clases Timer.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.



Detected stop



```
import java.util.Timer;
import java.util.TimerTask;

class CanStop extends Thread {
    private volatile boolean stop = false;
    private int counter = 0;

    public void run() {
        while (!stop && counter < 10000) {
            System.out.println(counter++);
        }
        if (stop)
            System.out.println("Detected stop");
    }
    public void requestStop() {
        stop = true;
    }
}
public class Stopping {
    public static void main(String[] args) {
        final CanStop stoppable = new CanStop();
        stoppable.start();

        new Timer(true).schedule(new TimerTask() {
            public void run() {
                System.out.println("Requesting stop");
                stoppable.requestStop();
            }
        },
        350);
    }
}
```

Ejemplos de Java: suspensión de un hilo

Descripción del problema

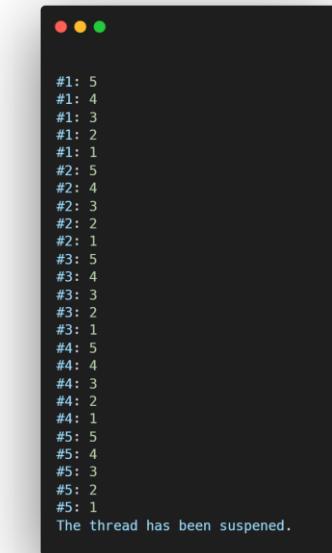
¿Cómo suspender un hilo por un tiempo?

Solución

El siguiente ejemplo muestra cómo suspender un hilo por un tiempo creando el método sleepingThread () .

Resultado

El ejemplo de código anterior producirá el siguiente resultado.



```
#1: 5
#1: 4
#1: 3
#1: 2
#1: 1
#2: 5
#2: 4
#2: 3
#2: 2
#2: 1
#3: 5
#3: 4
#3: 3
#3: 2
#3: 1
#4: 5
#4: 4
#4: 3
#4: 2
#4: 1
#5: 5
#5: 4
#5: 3
#5: 2
#5: 1
The thread has been suspened.
```

```
public class SleepingThread extends Thread {
    private int countDown = 5;
    private static int threadCount = 0;

    public SleepingThread() {
        super(" " + ++threadCount);
        start();
    }

    public String toString() {
        return "#" + getName() + ": " + countDown;
    }

    public void run() {
        while (true) {
            System.out.println(this);
            if (--countDown == 0) return;
            try {
                sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        for (int i = 0; i < 5; i++) new SleepingThread().join();
        System.out.println("The thread has been suspened.");
    }
}
```

Ejemplos de Java: ID de un hilo

Descripción del problema

¿Cómo obtener la identificación del hilo en ejecución?

Solución

El siguiente ejemplo demuestra cómo obtener el Id. De un hilo en ejecución usando el método `getThreadId()`.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.

```
threadA: in initialValue()
threadA: var getThreadId = 10001
threadB: in initialValue()
threadB: var getThreadId = 10002
threadC: in initialValue()
threadC: var getThreadId = 10003
threadA: var getThreadId = 10001
threadB: var getThreadId = 10002
threadC: var getThreadId = 10003
```

```
public class Main extends Object implements Runnable {
    private ThreadID var;
    public Main(ThreadID v) {
        this.var = v;
    }
    public void run() {
        try {
            print("var getThreadId =" + var.getThreadId());
            Thread.sleep(2000);
            print("var getThreadId =" + var.getThreadId());
        } catch (InterruptedException x) {}
    }
    private static void print(String msg) {
        String name = Thread.currentThread().getName();
        System.out.println(name + ": " + msg);
    }
    public static void main(String[] args) {
        ThreadID tid = new ThreadID();
        Main shared = new Main(tid);
        try {
            Thread threadA = new Thread(shared, "threadA");
            threadA.start();
            Thread.sleep(500);

            Thread threadB = new Thread(shared, "threadB");
            threadB.start();
            Thread.sleep(500);

            Thread threadC = new Thread(shared, "threadC");
            threadC.start();
        } catch (InterruptedException x) {}
    }
}
class ThreadID extends ThreadLocal {
    private int nextID;
    public ThreadID() {
        nextID = 10001;
    }
    private synchronized Integer getNewID() {
        Integer id = new Integer(nextID);
        nextID++;
        return id;
    }
    protected Object initialValue() {
        print("in initialValue()");
        return getNewID();
    }
    public int getThreadId() {
        Integer id = (Integer) get();
        return id.intValue();
    }
    private static void print(String msg) {
        String name = Thread.currentThread().getName();
        System.out.println(name + ": " + msg);
    }
}
```

Ejemplos de Java: comprobación de un hilo

Descripción del problema

¿Cómo comprobar el nivel de prioridad de un hilo?

Solución

El siguiente ejemplo demuestra cómo verificar el nivel de prioridad de un hilo usando el método `getPriority()` de un hilo.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.



```
in main() - Thread.currentThread().getPriority() = 5
in main() - Thread.currentThread().getName() = main
in run() - priority = 5, name = threadA
in run() - priority = 5, name = threadA
in main() - threadA.getPriority() = 5
in run() - priority = 5, name = threadA
in run() - priority = 5, name = threadA
in run() - priority = 5, name = threadA
```

```
public class Main extends Object {
    private static Runnable makeRunnable() {
        Runnable r = new Runnable() {
            public void run() {
                for (int i = 0; i < 5; i++) {
                    Thread t = Thread.currentThread();
                    System.out.println("in run() - priority = " + t.getPriority() + ", name = " + t.getName());
                    try {
                        Thread.sleep(2000);
                    } catch (InterruptedException x) {}
                }
            }
        };
        return r;
    }
    public static void main(String[] args) {
        System.out.println(
            "in main() - Thread.currentThread().getPriority() = " + Thread.currentThread().getPriority());
        System.out.println(
            "in main() - Thread.currentThread().getName() = " + Thread.currentThread().getName());

        Thread threadA = new Thread(makeRunnable(), "threadA");
        threadA.start();
        try {
            Thread.sleep(3000);
        } catch (InterruptedException x) {}
        System.out.println("in main() - threadA.getPriority() = " + threadA.getPriority());
    }
}
```

Ejemplos de Java: ¿mostrar todos los subprocessos en ejecución?

Descripción del problema

¿Cómo mostrar todos los hilos en ejecución?

Solución

El siguiente ejemplo demuestra cómo mostrar los nombres de todos los subprocessos en ejecución utilizando el método getName () .

```
public class Main extends Thread {  
    public static void main(String[] args) {  
        Main t1 = new Main();  
        t1.setName("thread1");  
        t1.start();  
        ThreadGroup currentGroup = Thread.currentThread().getThreadGroup();  
        int noThreads = currentGroup.activeCount();  
        Thread[] lstThreads = new Thread[noThreads];  
        currentGroup.enumerate(lstThreads);  
  
        for (int i = 0; i < noThreads; i++) System.out.println("Thread No:" + i + " = " + lstThreads[i].getName());  
    }  
}
```

Resultado

El ejemplo de código anterior producirá el siguiente resultado.

```
Thread No:0 = main  
Thread No:1 = thread1
```

Ejemplos de Java: mostrar el estado del hilo

Descripción del problema

¿Cómo mostrar el estado del hilo?

Solución

El siguiente ejemplo demuestra cómo mostrar diferentes estados del hilo usando los métodos `isAlive()` y `getstatus()` del hilo.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.

```
MyThread #1 Alive:false State:NEW
MyThread #1 starting.
waiting:true
waiting:true
alive
alive
MyThread #1 terminating.
alive
MyThread #1 Alive:false State:TERMINATED
```

```
class MyThread extends Thread {
    boolean waiting = true;
    boolean ready = false;
    MyThread() {
    }
    public void run() {
        String thrdName = Thread.currentThread().getName();
        System.out.println(thrdName + " starting.");

        while(waiting) System.out.println("waiting:" + waiting);
        System.out.println("waiting...");
        startWait();
        try {
            Thread.sleep(1000);
        } catch(Exception exc) {
            System.out.println(thrdName + " interrupted.");
        }
        System.out.println(thrdName + " terminating.");
    }
    synchronized void startWait() {
        try {
            while(!ready) wait();
        } catch(InterruptedException exc) {
            System.out.println("wait() interrupted");
        }
    }
    synchronized void notice() {
        ready = true;
        notify();
    }
}
public class Main {
    public static void main(String args[]) throws Exception {
        MyThread thrd = new MyThread();
        thrd.setName("MyThread #1");
        showThreadStatus(thrd);

        thrd.start();
        Thread.sleep(50);
        showThreadStatus(thrd);

        thrd.waiting = false;
        Thread.sleep(50);
        showThreadStatus(thrd);

        thrd.notice();
        Thread.sleep(50);
        showThreadStatus(thrd);

        while(thrd.isAlive())
            System.out.println("alive");
        showThreadStatus(thrd);
    }
    static void showThreadStatus(Thread thrd) {
        System.out.println(thrd.getName() + " Alive:" + thrd.isAlive() + " State:" + thrd.getState());
    }
}
```

Ejemplos de Java: interrumpir un hilo

Descripción del problema

¿Cómo interrumpir un hilo en ejecución?

Solución

El siguiente ejemplo demuestra cómo interrumpir un hilo en ejecución `interrupt()` método de hilo y comprobar si un hilo se interrumpe usando el método `isInterrupted()`.

Resultado

El ejemplo de código anterior producirá el siguiente resultado.



```
in run() - about to work2()
in main() - interrupting other thread
in main() - leaving
C isInterrupted() = true
in run() - interrupted in work2()
```

```
public class GeneralInterrupt extends Object implements Runnable {
    public void run() {
        try {
            System.out.println("in run() - about to work2()");
            work2();
            System.out.println("in run() - back from work2()");
        } catch (InterruptedException x) {
            System.out.println("in run() - interrupted in work2()");
            return;
        }
        System.out.println("in run() - doing stuff after nap");
        System.out.println("in run() - leaving normally");
    }
    public void work2() throws InterruptedException {
        while (true) {
            if (Thread.currentThread().isInterrupted()) {
                System.out.println("C isInterrupted()=" + Thread.currentThread().isInterrupted());
                Thread.sleep(2000);
                System.out.println("D isInterrupted()=" + Thread.currentThread().isInterrupted());
            }
        }
    }
    public void work() throws InterruptedException {
        while (true) {
            for (int i = 0; i < 100000; i++) {
                int j = i * 2;
            }
            System.out.println("A isInterrupted()=" + Thread.currentThread().isInterrupted());
            if (Thread.interrupted()) {
                System.out.println("B isInterrupted()=" + Thread.currentThread().isInterrupted());
                throw new InterruptedException();
            }
        }
    }
    public static void main(String[] args) {
        GeneralInterrupt si = new GeneralInterrupt();
        Thread t = new Thread(si);
        t.start();
        try {
            Thread.sleep(2000);
        } catch (InterruptedException x) { }

        System.out.println("in main() - interrupting other thread");
        t.interrupt();
        System.out.println("in main() - leaving");
    }
}
```