Planificación De Threads

Un hilo demonio en Java es un hilo proveedor de servicios que proporciona servicios al hilo de usuario.

Su vida depende de los demas hilos, es decir si todos los demas hilos se mueren la JVM lo termina automáticamente.

Hay muchos hilos demonios en java que se ejecutan automáticamente. Garbage Collector, finalizer, etc.

JConsole en el simbolo del sistema, proporciona la información sobre las clases cargadas, uso de memoria, ejecución de hilos etc.

Puntos importantes

Un hilo en Java proporciona servicios a los hilos para brindar soporte a tareas en segundo plano.

Su vida depende de los hilos de usuario. Es un hilo de baja prioridad.

Java.lang.Thread: Provee dos métodos para el demonio:

1. ***public setDaemon(boolean status):*** Es usado para marcar el hilo actual, como un hilo demonio.
2. ***public boolean isDaemon():*** Revisa si el hilo actual es un demonio.

```
package javaadvanced.Jueves;

import static java.lang.Thread.MAX_PRIORITY;
import static java.lang.Thread.MIN_PRIORITY;
import static java.lang.Thread.NORM_PRIORITY;

/**
 *
 * @author Ramses Santos
 */
public class ThreadD extends Thread{

    @Override
    public void run(){
        if(Thread.currentThread().isDaemon()){
            System.out.println("Tenemos un demonio aqui.");
        }
        else{
            System.out.println("Ah no es un hilo de usuario.");
        }
    }

    public static void main(String[] args) {
        ThreadD t1 = new ThreadD(), t2 = new ThreadD(), t3 = new ThreadD();

        t1.setDaemon(true);
        t1.start();
```

```
        t2.start();
        t3.start();
    }
}
```

Sincronización

La sincronización en JAVA es la capacidad de controlar el acceso de multiples hilos en cualquier recurso compartido.

La sincronización es la mejor opción cuando queremos permitir que solo un hilo pueda acceder a recursos compartidos.

La sincronización es principalmente usada para:

1. Prevenir la interferencia de hilos.
2. Prevenir problemas de inconsistencia.

Tipos de Sincronización:

1. Procesos
2. Hilos: Mutuamente Excluyentes
    1. Método Sincronizado.
    2. Bloqueo Sincronizado.
    3. Sincronización Estática.

Cooperación: Comunicación entre hilos. Exclusión Mutua: Ayuda a prevenir que los hilos interfieran entre si mientras comparten información.

Bloqueo: La sincronización se basa en una entidad interna conocida como lick o monito. Cada objeto tiene un bloque asociado a el.

Por convención un hilo necesita un acceso consistente a los campos de un objeto, tiene que adquirir el bloqueo del objeto antes de acceder a ellos y a continuación liberar el bloqueo cuando se hace con ellos.

Desde JAVA 5, el paquete ***java.util.concurrent.locks*** contiene varias implementaciones de bloqueo.

OutputStream vs InputStream

Java I/O

Entrada y Salida.

Se utiliza para procesar la entrada y salida. En JAVA se utiliza el concepto Stream.

El paquete ***java.io*** contiene todas las clases necesarias para realizar operaciones entrada y salida.

Podemos realizar el manejo de archivos a traves de JAVA I/O API.

Stream

Es un flujo de datos, que esta compuesto por bytes. Se llama stream porque es como un flujo de agua que sigue corriendo.

En JAVA existen 3 streams que se crean para nuestro uso automaticamente.

1. **System.out:** Flujo de salida estandar.
2. **System.in:** Flujo de entrada estandar.
3. **System.err:** Flujo de errores estandar.

```
System.out.print("Error");
System.err.print("Error");
```

OutputStream vs InputStream

Input: Lee.

Output: Escribe.

BufferedOutputStream

La clase JAVA BufferedOutputStream se utiliza para almacenar en un buffer un stream de salida.

Internamente utiliza un buffer para almacenar datos directamente en un Stream.

Añade mas eficiencia que escribir datos de manera directa en un flujo.

Por lo tanto logra un mejor rendimiento.

Tiene 2 constructores:

BufferedOutputStream(OutputStream os):

Crea un nuevo flujo de salida almacenando en buffer. Que se utiliza para escribir los datos en el stream especificado.

BufferedOutputStream(OutputStream os, int size):

Crea un nuevo flujo de salida que será almacenado en buffer el cual se utiliza para escribir los datos en el stream especificando el tamaño del bufer.

Métodos:

void write(int b):

Escribe el byte especificando en el stream de salida almacenado en buffer.

void write(byte[] b, int off, int len):

Escribe los bytes de la secuencia de stream de bytes especificada en un arreglo de bytes. Comienza con el desplazamiento dado.

void flush():

Borra el flujo de salida almacenado en buffer. Vacía los datos de una secuencia y la envía a otra. Se necesita si se ha conectado una secuencia con otra.

Repaso

- Callbacks
- Propiedades
- MultiTask != MultiHilo
- Clases Anonimas
- Syncronized
- Java.IO
- FileInputStream
- FileOutputStream
- BufferedInputStream
- BufferedOutputStream
- SequenceInputStream

Tarea

- Abstracción funcional, contextual, parametrica y de datos { concierto }
- Comic
- Networking.- (Solo Mapa Conceptual)
    - Mapa conceptual (No resumen)
    - TCP/IP
    - Datagramas
    - Sockets
    - Cliente/Servidor
    - Protocolos
    - Connection_Oriented
    - Connection_Less

- Peticiones/Get/Post
- URL
- Http/Https

Repaso

- Ip Address

```
asdasdasdasdasdas
 asdasd
 asdasd
```

- Protocol
- Port Number
- MAC Address
- Connection-Oriented and Connection-Less Protocol
- Socket

DatagramPacket y DatagramSocket