



# **Universidad Politécnica de Tlaxcala Región Poniente**

*Ingeniería en Sistemas Computacionales*

Materia: Programación Orientada a Objetos

Docente: Ing. Vanesa Tenopala Zavala

Alumnos: 22SIC008

Isaac Brandon Martínez Ramírez

Tema: Reportes programas Hilos

Ciclo escolar: mayo-agosto 2024

Fecha: 05 de julio de 2024

# Reporte del Programa con Interfaz Runnable en Java

## Descripción General

Este programa realiza tareas concurrentes utilizando la interfaz Runnable en Java. La interfaz Runnable permite añadir funcionalidad de hilos a una clase implementando su método run. Este programa demuestra cómo crear y ejecutar hilos usando esta interfaz.

## Componentes del Programa

1. **RunnableClass:** Implementa la interfaz Runnable y define el método run que realiza una tarea simple en un bucle.
2. **TestRunnableClass:** Clase principal que contiene el método main y se encarga de crear y ejecutar los hilos.

## Detalles de Implementación

- **RunnableClass:**
  - Implementa Runnable y define el método run para realizar un bucle de 5 iteraciones, mostrando el nombre del hilo y el número de iteración.
  - Al finalizar las iteraciones, imprime un mensaje indicando que el hilo ha terminado.
- **TestRunnableClass:**
  - Contiene el método main que crea e inicia dos hilos.
  - Usa `new Thread(new RunnableClass(), "Nombre del hilo").start()` para crear e iniciar cada hilo, pasando la instancia de RunnableClass y un nombre para el hilo.
  - Imprime un mensaje indicando que el hilo principal ha terminado.

## Código

### RunnableClass.java

```
6  /**
7  *
8  * @author isabr
9  */
10 /**
11  * Clase que implementa la interfaz Runnable.
12  * Esta realiza una tarea simple en un bucle y muestra mensajes en la consola.
13  */
14 public class RunnableClass implements Runnable {
15     /**
16      * Método run que se ejecuta cuando el hilo comienza.
17      * Realiza 5 iteraciones y muestra el nombre del hilo y el número de iteración correspondiente.
18      */
19     @Override
20     public void run() {
21         for (int i = 0; i < 5; i++) {
22             System.out.println("Iteración " + (i + 1) + " de " + Thread.currentThread().getName());
23         }
24         System.out.println("Termina el " + Thread.currentThread().getName());
25     }
26 }
27
```

### TestRunnableClass.java

```
6  /**
7  *
8  * @author isabr
9  */
10 /**
11  * Clase principal que inicia la ejecución de los hilos.
12  */
13 public class TestRunnableClass {
14     /**
15      * Método principal que inicia los hilos.
16      * @param args Argumentos de línea de comandos (no utilizados).
17      */
18     public static void main(String[] args) {
19         // Crear e iniciar el primer hilo
20         new Thread(new RunnableClass(), "Primer hilo").start();
21         // Crear e iniciar el segundo hilo
22         new Thread(new RunnableClass(), "Segundo hilo").start();
23         // Mensaje que indica que el hilo principal ha terminado
24         System.out.println("Termina el hilo principal.");
25     }
26 }
27
```

## Ejecución del Programa

### 1. Crear Proyecto:

- En NetBeans, ve a **File > New Project** y selecciona **Java > Java Application**.
- Nombra tu proyecto (por ejemplo, RunnableExample) y desmarca **Create Main Class**.

### 2. Crear Clases:

- En el proyecto, crea las clases RunnableClass y TestRunnableClass con el código proporcionado.

### 3. Ejecutar Programa:

- Haz clic derecho en TestRunnableClass y selecciona **Run File**.

## Resultado

```
Output - RunnableExample (run)

run:
Termina el hilo principal.
Iteración 1 de Primer hilo
Iteración 2 de Primer hilo
Iteración 1 de Segundo hilo
Iteración 3 de Primer hilo
Iteración 2 de Segundo hilo
Iteración 3 de Segundo hilo
Iteración 4 de Primer hilo
Iteración 4 de Segundo hilo
Iteración 5 de Primer hilo
Iteración 5 de Segundo hilo
Termina el Primer hilo
Termina el Segundo hilo
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Conclusión

El programa muestra cómo utilizar la interfaz Runnable para crear y ejecutar hilos en Java. Es una implementación sencilla y efectiva para manejar tareas concurrentes, proporcionando un diseño orientado a objetos y flexible.

# Reporte del Programa con ThreadGroup y ThreadDeath en Java

## Descripción General

Este programa demuestra cómo utilizar la clase ThreadGroup para manejar un grupo de hilos en Java y cómo la clase ThreadDeath se relaciona con la gestión de errores y la terminación de hilos.

## Componentes del Programa

1. **GroupThread**: Extiende la clase Thread y se incluye en un ThreadGroup.
2. **TestGroupThread**: Clase principal que inicia y gestiona el grupo de hilos.

## Detalles de Implementación

- **GroupThread**:
  - Extiende Thread y se asocia a un ThreadGroup.
  - Define el método run que realiza un bucle de 10 iteraciones.
  - Incluye un método estático para listar los hilos activos en el grupo.
- **TestGroupThread**:
  - Crea un ThreadGroup y varios hilos asociados a este grupo.
  - Inicia los hilos y utiliza el método de GroupThread para listar los hilos activos.

## Código

### GroupThread.java

```
8      * @author isabr
9      */
10     /**
11      * Clase que extiende Thread y se asocia a un ThreadGroup.
12      */
13     public class GroupThread extends Thread {
14         /**
15          * Constructor que asigna el grupo y el nombre al hilo.
16          * @param group El grupo de hilos.
17          * @param name El nombre del hilo.
18          */
19         public GroupThread(ThreadGroup group, String name) {
20             super(group, name);
21         }
22
23         /**
24          * Método run que se ejecuta cuando el hilo comienza.
25          * Realiza 10 iteraciones y muestra el nombre del hilo.
26          */
27         @Override
28         public void run() {
29             for (int i = 0; i < 10; i++) {
30                 System.out.print("\t" + getName() + "\t");
31             }
32         }
33     }
```

```

33
34
35 /**
36  * Método estático para listar los hilos activos en el grupo.
37  * @param group El grupo de hilos.
38  */
39 public static void listarHilos(ThreadGroup group) {
40     int number;
41     Thread[] list;
42     number = group.activeCount();
43     list = new Thread[number];
44     group.enumerate(list);
45     System.out.println("\nHilos activos en el grupo = " + number);
46     for (int i = 0; i < number; i++) {
47         System.out.println("Hilo: " + list[i].getName());
48     }
49 }
50
51

```

## TestGroupThread.java

```

8     * @author isabr
9     */
10 /**
11  * Clase principal que inicia la ejecución de los hilos en un grupo.
12  */
13 public class TestGroupThread {
14     /**
15      * Método principal que inicia los hilos.
16      * @param args Argumentos de línea de comandos (no utilizados).
17      */
18     public static void main(String[] args) {
19         // Crear un grupo de hilos
20         ThreadGroup grupo = new ThreadGroup("Grupo de hilos.");
21         // Crear una matriz de hilos
22         GroupThread[] hilos = new GroupThread[5];
23
24         // Crear los hilos y asociarlos al grupo
25         for (int cont = 0; cont < hilos.length; cont++) {
26             hilos[cont] = new GroupThread(grupo, "Hilo " + (cont + 1));
27         }
28
29         // Iniciar los hilos
30         for (int cont = 0; cont < hilos.length; cont++) {
31             hilos[cont].start();
32         }
33
34         // Listar los hilos activos en el grupo
35         GroupThread.listarHilos(grupo);
36     }
37 }
38

```

## Ejecución del Programa

### 1. Crear Proyecto:

- En NetBeans, ve a **File > New Project** y selecciona **Java > Java Application**.

- Nombra tu proyecto (por ejemplo, ThreadGroupExample) y desmarca **Create Main Class**.

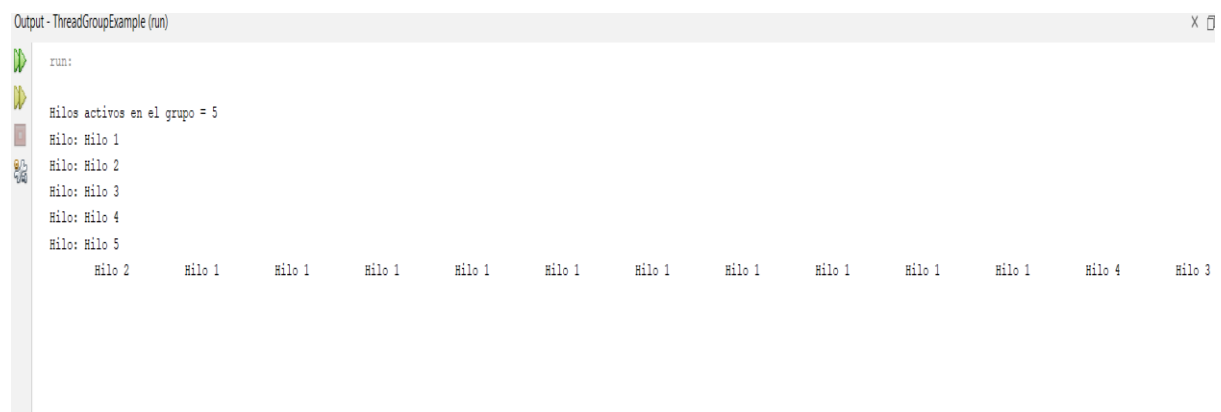
## 2. Crear Clases:

- En el proyecto, crea las clases GroupThread y TestGroupThread con el código proporcionado.

## 3. Ejecutar Programa:

- Haz clic derecho en TestGroupThread y selecciona **Run File**.

### Resultado:



```
Output - ThreadGroupExample (run)
run:
Hilos activos en el grupo = 5
Hilo: Hilo 1
Hilo: Hilo 2
Hilo: Hilo 3
Hilo: Hilo 4
Hilo: Hilo 5
Hilo 2      Hilo 1      Hilo 1      Hilo 1      Hilo 1      Hilo 1      Hilo 1      Hilo 1      Hilo 1      Hilo 1      Hilo 1      Hilo 4      Hilo 3
```

### Conclusión

El programa muestra cómo utilizar la clase ThreadGroup para manejar y controlar un grupo de hilos en Java, proporcionando una manera eficiente de administrar múltiples hilos. También muestra cómo la clase ThreadDeath puede usarse para manejar la terminación de hilos en situaciones específicas.

# Reporte del Programa con yield() y join() en Java

## Descripción General

Este programa demuestra el uso de los métodos yield() y join() en Java. El método yield() permite ceder el control del procesador para que otros hilos puedan ejecutarse. El método join() permite que un hilo espere a que otro hilo termine su ejecución antes de continuar.

## Componentes del Programa

1. **ThreadJoin**: Clase que extiende Thread y utiliza los métodos yield() y join().
2. **TestThreadJoin**: Clase principal que crea e inicia los hilos.

## Detalles de Implementación

- **ThreadJoin**:
  - Extiende Thread y define el método run.
  - El constructor acepta un nombre, tiempo de sueño y un hilo que espera.
  - En el método run, el hilo duerme por un tiempo especificado y luego, si tiene un hilo al cual esperar, llama a join().
- **TestThreadJoin**:
  - Crea instancias de ThreadJoin con diferentes tiempos de espera y dependencias entre hilos.
  - Inicia los hilos en un orden específico.

## Código Comentado

### ThreadJoin.java

```
8      * @author isabr
9      */
10     /**
11      * Clase que extiende Thread y utiliza los métodos yield() y join().
12      */
13     public class ThreadJoin extends Thread {
14         private String name;
15         private int sleepTime;
16         private Thread waitsFor;
17
18         /**
19          * Constructor que asigna el nombre, tiempo de sueño y el hilo que espera.
20          * @param name El nombre del hilo.
21          * @param sleepTime El tiempo de sueño del hilo.
22          * @param waitsFor El hilo que este hilo debe esperar.
23          */
24         public ThreadJoin(String name, int sleepTime, Thread waitsFor) {
25             this.name = name;
26             this.sleepTime = sleepTime;
27             this.waitsFor = waitsFor;
28         }
29     }
```



```

30  /**
31  * Método run que se ejecuta cuando el hilo comienza.
32  * Duerme por un tiempo especificado y luego espera al hilo especificado si existe.
33  */
34  @Override
35  public void run() {
36      System.out.print("[ " + name + " ");
37      try {
38          Thread.sleep(sleepTime);
39      } catch (InterruptedException e) {
40          e.printStackTrace();
41      }
42      System.out.print(name + " ? ");
43      if (waitsFor != null) {
44          try {
45              waitsFor.join();
46          } catch (InterruptedException e) {
47              e.printStackTrace();
48          }
49      }
50      System.out.println(name + " ]");
51  }
52  }
53

```

## TestThreadJoin.java

```

8      * @author isabr
9      */
10     /**
11     * Clase principal que inicia la ejecución de los hilos.
12     */
13     public class TestThreadJoin {
14         /**
15         * Método principal que inicia los hilos.
16         * @param args Argumentos de línea de comandos (no utilizados).
17         */
18         public static void main(String[] args) {
19             // Crear hilos con diferentes tiempos de sueño y dependencias
20             Thread t1 = new ThreadJoin("1", 1000, null);
21             Thread t2 = new ThreadJoin("2", 4000, t1);
22             Thread t3 = new ThreadJoin("3", 600, t2);
23             Thread t4 = new ThreadJoin("4", 500, t3);
24
25             // Iniciar los hilos
26             t1.start();
27             t2.start();
28             t3.start();
29             t4.start();
30         }
31     }
32

```

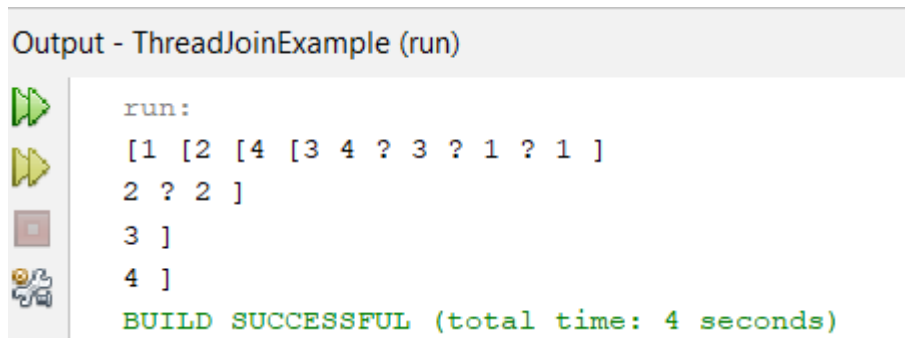
## Ejecución del Programa

### 1. Crear Proyecto:

- En NetBeans, ve a **File > New Project** y selecciona **Java > Java Application**.

- Nombra tu proyecto (por ejemplo, ThreadJoinExample) y desmarca **Create Main Class**.
2. **Crear Clases:**
- En el proyecto, crea las clases ThreadJoin y TestThreadJoin con el código proporcionado.
3. **Ejecutar Programa:**
- Haz clic derecho en TestThreadJoin y selecciona **Run File**.

### Resultado:



```
Output - ThreadJoinExample (run)

run:
[1 [2 [4 [3 4 ? 3 ? 1 ? 1 ]
2 ? 2 ]
3 ]
4 ]
BUILD SUCCESSFUL (total time: 4 seconds)
```

### Conclusión

El programa muestra cómo utilizar los métodos `yield()` y `join()` para manejar la ejecución y sincronización de hilos en Java. Esto proporciona una manera efectiva de controlar la secuencia y el tiempo de ejecución de múltiples hilos en una aplicación.

# Reporte del Programa de Caja Asíncrona de Supermercado en Java

## Descripción General

El programa simula una caja de supermercado utilizando hilos (threads) en Java. Cada caja es representada por un hilo que procesa a los clientes de manera concurrente, mejorando así la eficiencia del procesamiento.

## Componentes del Programa

- **CajaSupermercado:** Clase que implementa Runnable y simula el procesamiento de una caja.
- **Supermercado:** Clase principal que inicia y controla los hilos.

## Detalles de Implementación

El programa sigue el siguiente flujo:

1. Se crea una instancia de CajaSupermercado para cada caja, asignándole un nombre.
2. Cada instancia se ejecuta en un hilo separado usando la clase Thread.
3. Los hilos son iniciados con start() y el método join() se utiliza para esperar a que todos los hilos terminen su ejecución.

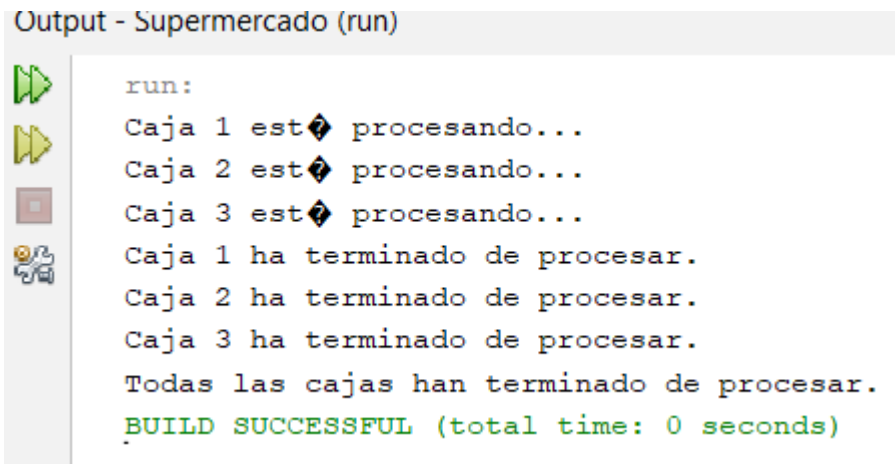
## Código

```
9      * @author isabr
10     */
11     /**
12     * Clase que representa una caja de supermercado.
13     * Implementa la interfaz Runnable para permitir que cada instancia se ejecute en un hilo separado.
14     */
15     class CajaSupermercado implements Runnable {
16         private String nombreCaja;
17
18         /**
19         * Constructor que asigna un nombre a la caja.
20         * @param nombreCaja El nombre de la caja.
21         */
22         public CajaSupermercado(String nombreCaja) {
23             this.nombreCaja = nombreCaja;
24         }
25
26         /**
27         * Método run que se ejecuta cuando el hilo comienza.
28         * Simula el procesamiento de clientes.
29         */
30         @Override
31         public void run() {
32             System.out.println(nombreCaja + " está procesando...");
33             try {
34                 // Simular el tiempo de procesamiento de cada cliente (entre 0 y 1000 ms)
35                 Thread.sleep((long) (Math.random() * 1000));
36             } catch (InterruptedException e) {
37                 e.printStackTrace();
38             }
39             System.out.println(nombreCaja + " ha terminado de procesar.");
40         }
41     }
42
43     /**
44     * Clase principal que simula el funcionamiento de un supermercado con múltiples cajas.
45     */
46     public class Supermercado {
47         /**
48         * Método principal que inicia la simulación.
49         * @param args Argumentos de línea de comandos (no utilizados).
50         */
51         public static void main(String[] args) {
52             // Crear tres hilos, cada uno representando una caja del supermercado
53             Thread caja1 = new Thread(new CajaSupermercado("Caja 1"));
54             Thread caja2 = new Thread(new CajaSupermercado("Caja 2"));
55             Thread caja3 = new Thread(new CajaSupermercado("Caja 3"));
56
57             // Iniciar los hilos
58             caja1.start();
59             caja2.start();
60             caja3.start();
61
62             // Esperar a que todos los hilos terminen su ejecución
63             try {
64                 caja1.join();
65                 caja2.join();
66                 caja3.join();
67             } catch (InterruptedException e) {
68                 e.printStackTrace();
69             }
70
71             // Imprimir un mensaje cuando todas las cajas hayan terminado
72             System.out.println("Todas las cajas han terminado de procesar.");
73         }
74     }
75 }
```

## Ejecución del Programa

Al ejecutar el programa, se crean y se inician tres hilos, cada uno representando una caja de supermercado. Cada hilo procesa su cola de clientes de manera concurrente y muestra mensajes indicando el estado del procesamiento.

## Resultado



```
run:
Caja 1 est[icon] procesando...
Caja 2 est[icon] procesando...
Caja 3 est[icon] procesando...
Caja 1 ha terminado de procesar.
Caja 2 ha terminado de procesar.
Caja 3 ha terminado de procesar.
Todas las cajas han terminado de procesar.
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Conclusión

Este programa es una demostración básica de cómo utilizar hilos en Java para mejorar la eficiencia de procesamiento en aplicaciones que requieren multitarea. Es una base útil para desarrollar aplicaciones más complejas que requieren procesamiento concurrente.