Threads – Java Concurrency Cookbook

Programación Concurrente 2017

Ing. Ventre, Luis O.

Waiting for the finalization

- En algunas situaciones, debemos esperar a que un Thread finalice su ejecución.
  - Por ejemplo: un programa que comience inicializando los recursos antes de comenzar con el resto de la ejecución.

- Para este propósito podemos usar el método join() de la clase Thread.
  - Cuando llamamos este método usando un objeto Thread, suspende su ejecución hasta que finalice la ejecución el Thread llamado.

Waiting for the finalization

- Veremos el ejemplo de inicialización:
  - Crear una clase llamada DataSourcesLoader, la misma debe implementar la interfaz runnable.
  - Implemente el método run(), este escribe un mensaje indicando que comienza su ejecución; luego se duerme por 4 segundos, y escribe otro mensaje para indicar que finalizó su ejecución.

  - Crear una clase llamada NetworkConnectionsLoader, la misma debe implementar la interfaz runnable.
  - El método run debe ser igual al anterior pero 6 segundos

Waiting for the finalization

- Crear la clase main.
  - Crear un objeto de cada clase anterior.
  - Crear dos Threads y lanzar la ejecución.

- Desde el main, deberá esperar la finalización de ambos hilos. Utilizando el método join(). Este método puede lanzar una InterruptedException por lo que se debe incluir try catch.

- Imprimir un mensaje de fin de programa. Ejecutar!

Waiting for the finalization

- La clase main

```
import java.util.Date;

public class Main {

    public static void main(String[] args) {
        DataSourcesLoader dsLoader = new DataSourcesLoader();
        Thread thread1 = new Thread(dsLoader,"DataSourceThread");//Tipo y nombre del thread creado

        NetworkConnectionsLoader ntLoader = new NetworkConnectionsLoader();
        Thread thread2 = new Thread(ntLoader,"NetworkConnectionsLoader Thread");

        thread1.start();
        thread2.start();

        try {
            thread1.join();   //es lo mismo que hacer join solo con el segundo
            thread2.join(); //aca el join hace que el main detenga su ejecucion hasta que finalize
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.printf("Main: Configuration has been loaded: %s\n",new Date());
    }

}
```

Waiting for the finalization

- La clase DataSourcesLoader

```
import java.util.Date;

public class DataSourcesLoader implements Runnable {

    @Override
    public void run() {
        System.out.printf("Beginning data sources loading: %s\n",new Date());
        try {
            TimeUnit.SECONDS.sleep(4);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.printf("Data sources loading has finished: %s\n", new Date());
    }

}
```

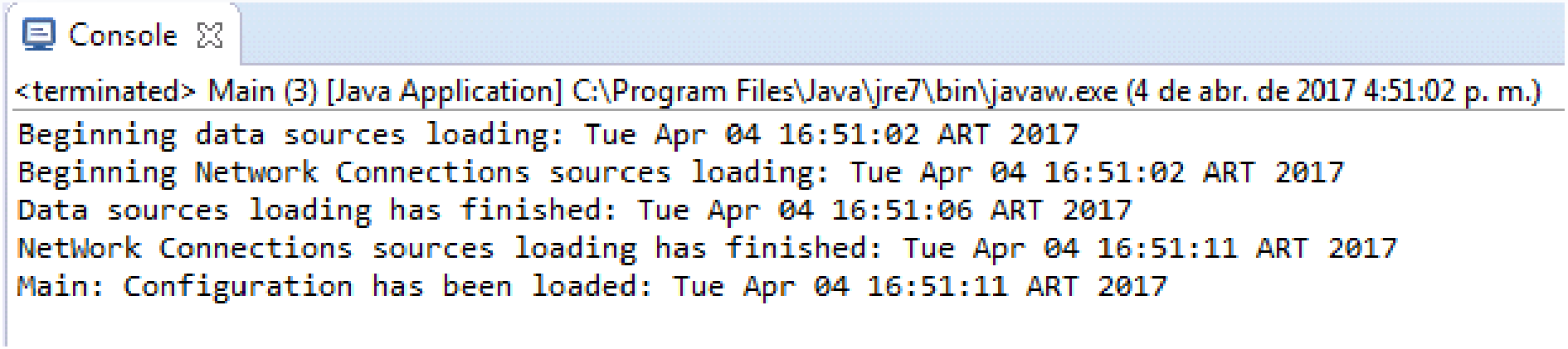
Waiting for the finalization

- La clase NetworkConnectionsLoader

```
import java.util.Date;

public class NetworkConnectionsLoader implements Runnable {

    @Override
    public void run() {
        System.out.printf("Beginning data sources loading: %s\n",new
                Date());
        try {
            TimeUnit.SECONDS.sleep(9);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.printf("Data sources loading has finished: %s\n",new Date());
    }

}
```

Waiting for the finalization

- La salida debe ser similar a:



Creating and running a Daemon Thread

- Java tiene un tipo especial de hilo “Daemon”:
  - Este tipo de hilos tiene una **muy baja prioridad de ejecución**. Normalmente ejecutan cuando ningun otro hilo del mismo programa está ejecutando.
  - Cuando los hilos daemons son solo los hilos vivos de un programa la JVM los finaliza.
  - Debido a estas características los hilos Daemons se utilizan para brindar servicios en segundo plano.
  - Generalmente no se utilizan para tareas importantes, por la incertidumbre de cuando tendrán tiempo de cpu.

Creating and running a Daemon Thread

- En el siguiente ejercicio se verá como implementar un hilo daemon.

- El objetivo del mismo son 3 hilos principales que escriben eventos en una cola y el hilo daemon será el encargado de ir limpiando los eventos de la cola que tienen mas de 10 segundos desde el tiempo de su creación.

Creating and running a Daemon Thread

- Crear la clase event.
  - Esta clase almacena solo información de los eventos. Tiene 2 atributos privados.
- Crear la clase WriterTask.
  - Implementar la interfaz runnable.
  - Declara la cola que almacena los eventos e implemente el constructor que inicializa la cola.
  - Implemente el método run. Que tendrá un loop de 100 iteraciones donde se crea un new Event, se guarda en la cola y se duerme por 1 segundo.

Creating and running a Daemon Thread

- Crear la clase CleanerTask.
  - Implementarla como extension de la clase Thread.
  - Declara la cola que almacena los eventos e implemente el constructor de la clase.
  - En el constructor marcar este hilo como Daemon con el metodo **setDaemon().**
  - Implementar el método run().
    - Un bucle infinito.
    - Ejecuta el método clean().

Creating and running a Daemon Thread

- Implementar el método Clean().
  - El mismo toma el ultimo evento, y si el mismo tiene mas de 10 segundos desde su momento de creación, es eliminado.
  - Y se continúa chequeando el siguiente evento.
  - Si un evento es borrado, se muestra en la consola y el tamaño resultante de la cola.
- Implementar la clase Main.
  - Crear la cola .
  - Crear e inicializar 3 WriterTask y 1 Cleaner Task.
  - Lanzar el programa y ver los resultados.

Creating and running a Daemon Thread

- Clase Event

```
package com.packtpub.java7.concurrency.chapter1.recipe7.event;

import java.util.Date;

public class Event {

    private Date date;
    private String event;

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getEvent() {
        return event;
    }

    public void setEvent(String event) {
        this.event = event;
    }
}
```

Creating and running a Daemon Thread

- Clase WriterTask

```java
package com.packtpub.java7.concurrency.chapter1.recipe7.task;

import java.util.Date;

public class WriterTask implements Runnable {

    Deque<Event> deque;

    public WriterTask (Deque<Event> deque){
        this.deque=deque;
    }

    @Override
    public void run() {

        // Writes 100 events
        for (int i=1; i<100; i++) {
            // Creates and initializes the Event objects
            Event event=new Event();
            event.setDate(new Date());
            event.setEvent(String.format("The thread %s has generated an event",
                    Thread.currentThread().getId()));

            // Add to the data structure
            deque.addFirst(event);
            try {
                // Sleeps during one second
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Creating and running a Daemon Thread

- Clase CleanerTask

```
package com.packtpub.java7.concurrency.chapter1.recipe7.task;

import java.util.Date;

public class CleanerTask extends Thread {

    private Deque<Event> deque;

    public CleanerTask(Deque<Event> deque) {
        this.deque = deque;
        // Establish that this is a Daemon Thread
        setDaemon(true);
    }

    @Override
    public void run() {
        while (true) {
            Date date = new Date();
            clean(date);
        }
    }
  private void clean(Date date) {
      long difference;
      boolean delete;

      if (deque.size()==0) {
          return;
      }

      delete=false;
      do {
          Event e = deque.getLast();
          difference = date.getTime() - e.getDate().getTime();
          if (difference > 10000) {
              System.out.printf("Cleaner: %s\n",e.getEvent());
              deque.removeLast();
              delete=true;
          }
      } while (difference > 10000);
      if (delete){
          System.out.printf("Cleaner: Size of the queue: %d\n",deque.size());
      }
  }
```

Creating and running a Daemon Thread

- Clase Main

```
package com.packtpub.java7.concurrency.chapter1.recipe7.core;

import java.util.ArrayDeque;

public class Main {

    public static void main(String[] args) {

        // Creates the Event data structure
        Deque<Event> deque=new ArrayDeque<Event>();

        // Creates the three WriterTask and starts them
        WriterTask writer=new WriterTask(deque);
        for (int i=0; i<3; i++){
            Thread thread=new Thread(writer);
            thread.start();
        }

        // Creates a cleaner task and starts them
        CleanerTask cleaner=new CleanerTask(deque);
        cleaner.start();

    }

}
```

Creating and running a Daemon Thread

- Como funciona:
- Si analizamos una ejecución puede observarse como la cola comienza a crecer hasta que alcanza 30 eventos. Su tamaño varia entre 27 y 30.
- El programa arranca con 3 hilos escritores. Cada uno escribe un evento y se duerme por un segundo.
- Luego de los primero 10 segundos, tenemos 30 eventos en la cola.

Creating and running a Daemon Thread

- **Luego de estos 10 segundos, el hilo daemon se ejecutaba mientras los hilos escritores dormían, pero no borro ninguno porque ninguno tenía mas de 10 segundos de creado.**
- Durante el resto de la ejecución el cleaner elimina 3 eventos por segundo y los 3 hilos crean 1 cada uno. Por lo tanto la cola mantiene su tamaño.

- Puede jugar con el tiempo de creado de los eventos para ser eliminados y el resultado?

Creating and running a Daemon Thread

- Algunos detalles:
  - El método setDaemon, solo puede ejecutarse ANTES de inicializar el hilo. Una vez ejecutado start, no puede modificarse su daemon status.
  - Se puede utilizar el método isDaemon, para consultar si el hilo es de este tipo. Este método devuelve True or False.

Using Local Thread Variables

- Uno de los aspectos mas críticos de las aplicaciones concurrentes son las variables compartidas.
- Esto tiene vital importancia en las clases que extienden la clase Thread o implementan la interfaz runnable.

- Si ud. crea un objeto de una clase que implementa la interfaz runnable, y luego crea varios hilos usando el mismo objeto runnable.

Using Local Thread Variables

- Todos los hilos comparten los mismo atributos.
- Si ud. cambia la variable en un hilo, afectará a todos.

- A veces ud necesita un atributo que no sea compartido por los hilos. Java provee un mecanismo llamado variables locales de thread.

- Veremos el problema y la solución.

Using Local Thread Variables

- Veremos un ejemplo del problema anterior:
- Cree una clase llamada UnsafeTask que implemente runnable.
  - Declarar un atributo privado de tipo date.
- Implemente el método run().
  - Este método inicializa el atributo startDate, imprime su valor por la consola, se duerme un número random de tiempo y vuelve a escribirlo.
- Implemente la clase main. Este método debe crear un objeto Unsafe Task y lanzar 3 hilos con igual argumento. (durmiendo 2 segundo entre creación)

Using Local Thread Variables

- Clase Main

```
package com.packtpub.java7.concurrency.chapter1.recipe7.core;

import java.util.concurrent.TimeUnit;

public class Main {

    public static void main(String[] args) {
        // Creates the unsafe task
        UnsafeTask task=new UnsafeTask();

        // Throw three Thread objects
        for (int i=0; i<10; i++){
            Thread thread=new Thread(task);
            thread.start();
            try {
                TimeUnit.SECONDS.sleep(2);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Using Local Thread Variables

- Clase UnsafeTask

```
package com.packtpub.java7.concurrency.chapter1.recipe7.task;

import java.util.Date;

public class UnsafeTask implements Runnable{

    private Date startDate;

    @Override
    public void run() {
        startDate=new Date();
        System.out.printf("Starting Thread: %s : %s\n",
                Thread.currentThread().getId(),startDate);
        try {
            TimeUnit.SECONDS.sleep((int)Math.rint(Math.random()*10));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.printf("Thread Finished: %s : %s\n",
                Thread.currentThread().getId(),startDate);
    }

}
```

Using Local Thread Variables

- El resultado de la ejecución muestra cada hilo tiene un tiempo de inicio diferente, pero el finalización es el mismo.

```
Starting Thread: 9 : Mon Apr 03 20:17:26 ART 2017
Starting Thread: 10 : Mon Apr 03 20:17:28 ART 2017
Starting Thread: 11 : Mon Apr 03 20:17:30 ART 2017
Starting Thread: 12 : Mon Apr 03 20:17:32 ART 2017
Starting Thread: 13 : Mon Apr 03 20:17:34 ART 2017
Thread Finished: 9 : Mon Apr 03 20:17:34 ART 2017
Thread Finished: 10 : Mon Apr 03 20:17:34 ART 2017
Starting Thread: 14 : Mon Apr 03 20:17:36 ART 2017
Thread Finished: 14 : Mon Apr 03 20:17:36 ART 2017
Starting Thread: 15 : Mon Apr 03 20:17:38 ART 2017
Thread Finished: 11 : Mon Apr 03 20:17:38 ART 2017
Thread Finished: 13 : Mon Apr 03 20:17:38 ART 2017
Starting Thread: 16 : Mon Apr 03 20:17:40 ART 2017
Thread Finished: 12 : Mon Apr 03 20:17:40 ART 2017
Starting Thread: 17 : Mon Apr 03 20:17:42 ART 2017
Starting Thread: 18 : Mon Apr 03 20:17:44 ART 2017
Thread Finished: 15 : Mon Apr 03 20:17:44 ART 2017
Thread Finished: 18 : Mon Apr 03 20:17:44 ART 2017
Thread Finished: 16 : Mon Apr 03 20:17:44 ART 2017
Thread Finished: 17 : Mon Apr 03 20:17:44 ART 2017
```

Using Local Thread Variables

- Como se mencionó anteriormente, para resolver esto se usaran variables locales de hilo.
- Cree una clase llamada SafeTask, que implemente runnable.
- Declare un objeto de la clase ThreadLocal<date>.
  - Este objeto tendra un implementacion implicita que incluye el metodo InitialValue. Este metodo devuelve el valor actual.
  - Implemente el método run(). Funciona similar al ejemplo anterior, solo cambia el acceso a la vble local.
  - Ejecute el ejemplo y COMPARE!

Using Local Thread Variables

- Clase SafeMain

```
package com.packtpub.java7.concurrency.chapter1.recipe7.core;

import java.util.concurrent.TimeUnit;

public class SafeMain {

    public static void main(String[] args) {
        // Creates a task
        SafeTask task=new SafeTask();

        // Creates and start three Thread objects for that Task
        for (int i=0; i<3; i++){
            Thread thread=new Thread(task);
            try {
                TimeUnit.SECONDS.sleep(2);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            thread.start();
        }

    }

}
```

Using Local Thread Variables

- Clase SafeTask

```
package com.packtpub.java7.concurrency.chapter1.recipe7.task;

import java.util.Date;
public class SafeTask implements Runnable {

    private static ThreadLocal<Date> startDate= new ThreadLocal<Date>() {
        protected Date initialValue(){
            return new Date();
        }
    };

    @Override
    public void run() {
        // Writes the start date
        System.out.printf("Starting Thread: %s : %s\n",
                Thread.currentThread().getId(),startDate.get());
        try {
            TimeUnit.SECONDS.sleep((int)Math.rint(Math.random()*10));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        // Writes the start date
        System.out.printf("Thread Finished: %s : %s\n",
                Thread.currentThread().getId(),startDate.get());
    }
```

Using Local Thread Variables

- La ejecución del proyecto con variables locales de hilo debería ser similar a:

```
Starting Thread: 9 : Mon Apr 03 20:32:19 ART 2017
Starting Thread: 10 : Mon Apr 03 20:32:21 ART 2017
Thread Finished: 9 : Mon Apr 03 20:32:19 ART 2017
Starting Thread: 11 : Mon Apr 03 20:32:23 ART 2017
Thread Finished: 11 : Mon Apr 03 20:32:23 ART 2017
Starting Thread: 12 : Mon Apr 03 20:32:25 ART 2017
Thread Finished: 10 : Mon Apr 03 20:32:21 ART 2017
Starting Thread: 13 : Mon Apr 03 20:32:27 ART 2017
Starting Thread: 14 : Mon Apr 03 20:32:29 ART 2017
Thread Finished: 13 : Mon Apr 03 20:32:27 ART 2017
Starting Thread: 15 : Mon Apr 03 20:32:31 ART 2017
Starting Thread: 16 : Mon Apr 03 20:32:33 ART 2017
Thread Finished: 12 : Mon Apr 03 20:32:25 ART 2017
Starting Thread: 17 : Mon Apr 03 20:32:35 ART 2017
Thread Finished: 14 : Mon Apr 03 20:32:29 ART 2017
Starting Thread: 18 : Mon Apr 03 20:32:37 ART 2017
Thread Finished: 17 : Mon Apr 03 20:32:35 ART 2017
Thread Finished: 15 : Mon Apr 03 20:32:31 ART 2017
Thread Finished: 16 : Mon Apr 03 20:32:33 ART 2017
Thread Finished: 18 : Mon Apr 03 20:32:37 ART 2017
```