Threads – Java Concurrency Cookbook

Programacion Concurrente 2017

Ing. Ventre, Luis O.

Creating and Running A Thread

- Aprenderemos como crear y ejecutar un Hilo en una aplicación Java.

- Los hilos (en inglés Threads) son objetos.
  - Hay dos formas de crear Hilos en JAVA:
    - Extendiendo la clase Thread y sobreescribiendo el método run().
    - Creando una clase que implemente la interfaz ***runnable***. Luego crear un objeto de la clase Thread y pasarle el objeto runnable como argumento.

Creating and Running Threads

- En el próximo ejercicio, usaremos la segunda forma de creación de hilos; para crear un programa simple que crea y ejecuta 10 hilos.
- Cada hilo calcula e imprime una tabla de multiplicación entre 1 y 10.
- El ejemplo se implementará en la Ide Eclipse.

Creating and Running - HOW

- Se implementará el proyecto paso a paso:

1. Crear una clase llamada Calculator, que implemente la interfaz “runnable”.

```
public class Calculator implements Runnable {
```

2. Declare un atributo privado int llamado “number”, e implemente el constructor de la clase que inicializa este valor.

```
private int number;

public Calculator(int number) {
    super();
    this.number = number;
}
```

Creating and Running - HOW

3. Implemente el método run(). Este método ejecutará las instrucciones de los threads que estamos creando.

```
public void run() {

  for (int i = 1; i <= 10; i++) {
    System.out.printf("%s: %d * %d = %d\n",
                Thread.currentThread().getName(), number, i, i * number);
  }
}
```

Creating and Running - HOW

3. Ahora se implementará la clase Main() de la aplicación.

```
public class Main {

    public static void main(String[] args) {
```

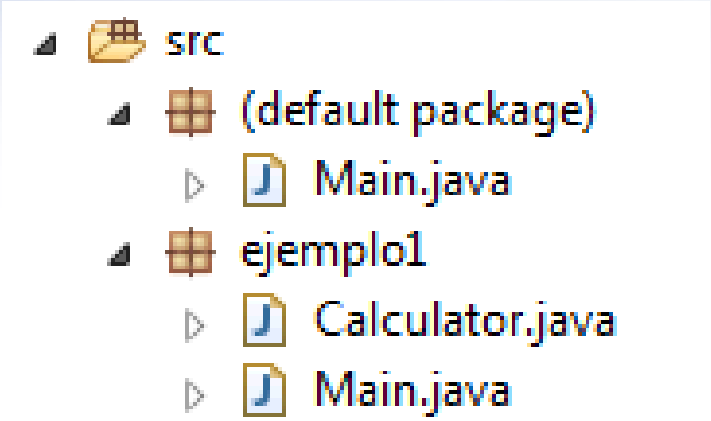
4. Adentro de la clase main, crear un bucle con 10 iteraciones. Adentro del loop crear un objeto de la clase Calculator, un objeto Thread, y llamar el metodo start.

```
for (int i = 1; i <= 10; i++) {
    Calculator calculator = new Calculator(i);
    Thread thread = new Thread(calculator);
    thread.start();
}
```

5. Ejecute el programa, y observe como los hilos trabajan en paralelo!

Creating and Running - HOW

6. La estructura de su proyecto debe asimilarse a la siguiente:



7. Y la salida de Consola:

```
Thread-9: 10 * 8 = 80
Thread-8: 9 * 9 = 81
Thread-4: 5 * 9 = 45
Thread-2: 3 * 10 = 30
Thread-0: 1 * 5 = 5
Thread-4: 5 * 10 = 50
Thread-8: 9 * 10 = 90
Thread-9: 10 * 9 = 90
Thread-0: 1 * 6 = 6
Thread-9: 10 * 10 = 100
Thread-0: 1 * 7 = 7
Thread-0: 1 * 8 = 8
Thread-0: 1 * 9 = 9
Thread-0: 1 * 10 = 10
```

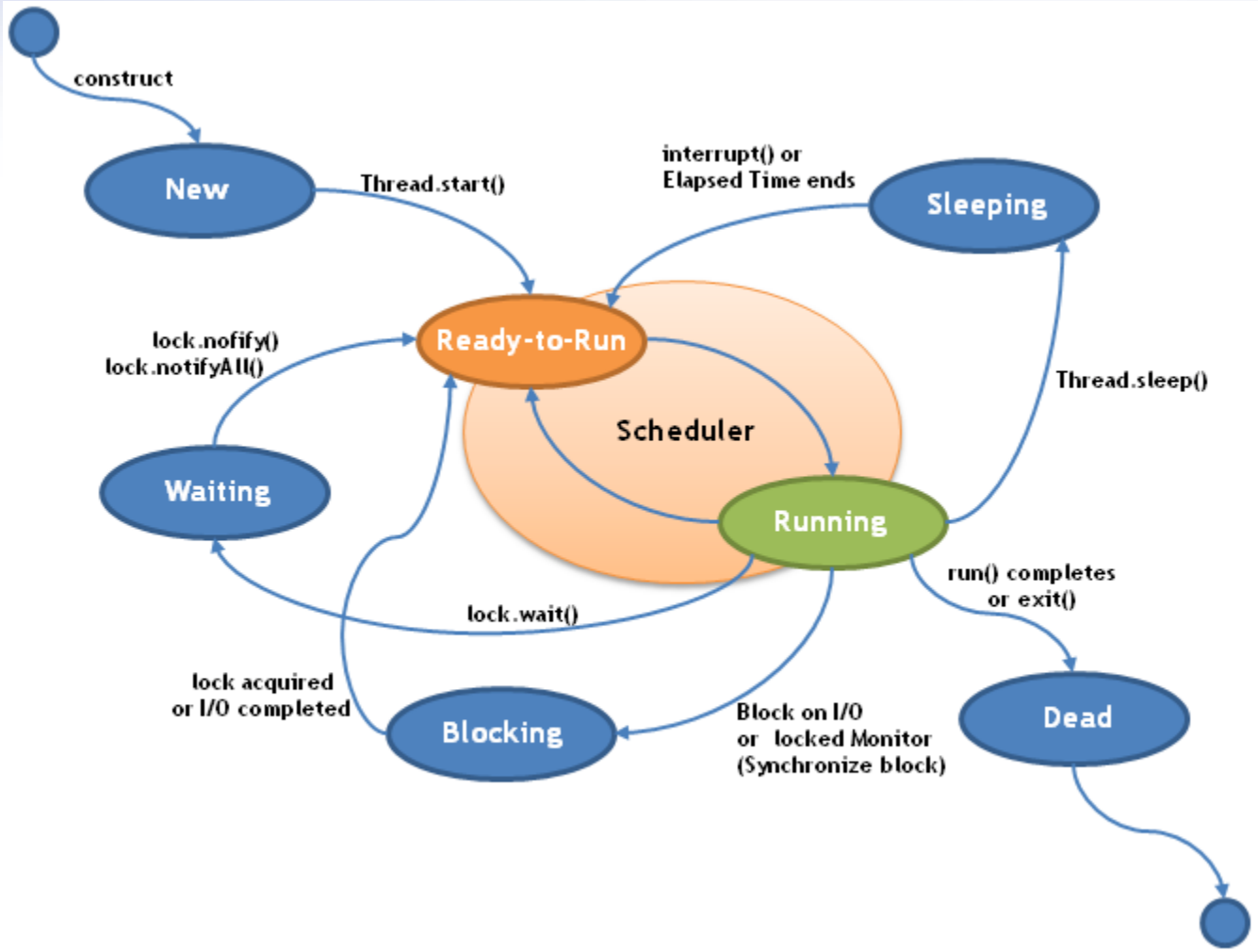
Getting & Setting Thread Info

- La clase Thread, almacena atributos de informacion que pueden ayudarnos a identificar un thread, conocer su estado, o controlar su prioridad. Estos atributos son:

  - ID: Este atributo almacena un identificador único para cada thread.
  - Nombre: Este atributo almacena el nombre del hilo.
  - Priority: Este atributo almacena la prioridad de los objetos hilo.
    - » La prioridad en los hilos varia desde 1 hasta 10. 1 es la mas baja. 10 la más alta. Cambiar la prioridad de los hilos no es recomendable.
  - Estado: Este atributo almacena el estado del hilo. En java un hilo puede tener uno de seis estados:
    - * New
    - * Ready to run(Runnable)
    - * Blocked
    - * Waiting
    - * Sleeping
    - * Terminated or Dead

Getting & Setting Thread Info

- Estado: Este atributo almacena el estado del hilo. En java un hilo puede tener uno de seis estados:



Getting & Setting Thread Info

1. El siguiente proyecto implementará un programa que establece el nombre y la prioridad de 10 threads, y muestra la info del estado hasta que finaliza. Los hilos calcularán la tabla de multiplicación.

2. El ejemplo ha sido implementado en la ide Java.

Getting & Setting Thread Info - HOW

- Se implementará el proyecto paso a paso:

1. Crear una clase llamada Calculator, que implemente la interfaz “runnable”.

```
public class Calculator implements Runnable {
```

2. Declare un atributo private int llamado “number”, e implemente el constructor de la clase que inicializa este valor.

```
private int number;

public Calculator(int number) {
    super();
    this.number = number;
}
```

Getting & Setting Thread Info - HOW

3. Implemente el método run(). Este método ejecutará las instrucciones de los threads que estamos creando.

```
public void run() {

  for (int i = 1; i <= 10; i++) {
    System.out.printf("%s: %d * %d = %d\n",
                Thread.currentThread().getName(), number, i, i * number);
  }
}
```

Getting & Setting Thread Info - HOW

4. Ahora se implementará la clase Main de este proyecto.

```
public class Main {

    public static void main(String[] args) {
```

5. Crear un array de 10 hilos, y un array de 10 estados de hilo para almacenar los hilos y estados

```
Thread threads[] = new Thread[10];
Thread.State status[] = new Thread.State[10];
```

Getting & Setting Thread Info - HOW

6. Crear 10 objetos de clase Calculator, cada uno inicializado con un número diferente; y 10 hilos para ejecutarlos. Setear prioridad máxima a 5 de ellos y mínima a 5 de ellos.

```
for (int i = 0; i < 10; i++) {
    threads[i] = new Thread(new Calculator(i));
    if ((i % 2) == 0) {
        threads[i].setPriority(Thread.MAX_PRIORITY);
    } else {
        threads[i].setPriority(Thread.MIN_PRIORITY);
    }
    threads[i].setName("Thread " + i);
}
```

Getting & Setting Thread Info - HOW

7. Crear un objeto PrintWriter para guardar en un archivo la evolución de los estados de los hilos.

```
try (FileWriter file = new FileWriter("C:/Concurrente/2016/log.txt");
PrintWriter pw = new PrintWriter(file);) {
```

8. Guardar en este archivo el estado de los hilos. Ahora son new.

```
for (int i = 0; i < 10; i++) {
  pw.println("Main : Status of Thread " + i + " : " + threads[i].getState());
  status[i] = threads[i].getState();
}
```

9. Start() la ejecución de los hilos

```
for (int i = 0; i < 10; i++) {
    threads[i].start();
}
```

Getting & Setting Thread Info - HOW

10. Hasta la finalización de los 10 hilos, vamos a chequear su estado. Si detectamos un cambio de estado en el hilo, se escribe en el archivo.

```
boolean finish = false;
while (!finish) {
    for (int i = 0; i < 10; i++) {
        if (threads[i].getState() != status[i]) {
            writeThreadInfo(pw, threads[i], status[i]);
            status[i] = threads[i].getState();
        }
    }
    finish = true;
    for (int i = 0; i < 10; i++) {
        finish = finish && (threads[i].getState() == State.TERMINATED);
    }
}
```

Getting & Setting Thread Info - HOW

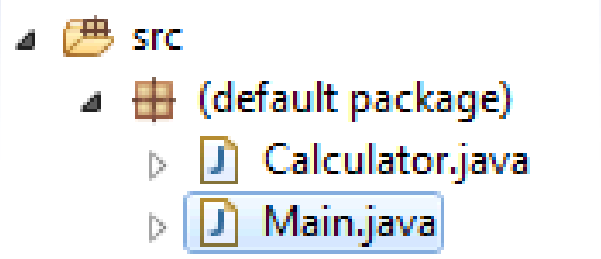
11. Implementar el método writeThreadInfo(), el cual escribe el ID, nombre, prioridad, estado anterior y nuevo.

```
private static void writeThreadInfo(PrintWriter pw, Thread thread, State state) {
    pw.printf("Main : Id %d - %s\n", thread.getId(), thread.getName());
    pw.printf("Main : Priority: %d\n", thread.getPriority());
    pw.printf("Main : Old State: %s\n", state);
    pw.printf("Main : New State: %s\n", thread.getState());
    pw.printf("Main : ************************************\n");
}
```

12. Ejecutar el ejemplo, y abrir el archivo “log.txt” para ver la evolución de los 10 hilos!.

Getting & Setting Thread Info - HOW

13. La estructura de su proyecto debe ser similar a la siguiente:



14. La salida por consola:

```
Main : Status of Thread 2 : NEW
Main : Status of Thread 3 : NEW
Main : Status of Thread 4 : NEW
Main : Status of Thread 5 : NEW
Main : Status of Thread 6 : NEW
Main : Status of Thread 7 : NEW
Main : Status of Thread 8 : NEW
Main : Status of Thread 9 : NEW
Main : Id 12 - Thread 0
Main : Priority: 10
Main : Old State: NEW
Main : New State: TERMINATED
Main : ************************************
Main : Id 13 - Thread 1
Main : Priority: 1
Main : Old State: NEW
Main : New State: TERMINATED
Main : ************************************
Main : Id 14 - Thread 2
Main : Priority: 10
Main : Old State: NEW
Main : New State: TERMINATED
Main : ************************************
Main : Id 15 - Thread 3
```

Dos hilos…

- A continuación se observara el código de ejemplo brindado en la primer clase Teórica.

- Puede Predecir el comportamiento del programa?
- Haga el debugg correspondiente analizando la traza.
- El comportamiento del programa es siempre el mismo?

Dos Hilos

Implemente un proyecto en la ide Eclipse con la siguiente estructura:

Clase Main

```
public class Main {

    public static void main(String[] args) {

        new Proceso();

    }

}
```

Dos Hilos

Clase Proceso

```
public class Proceso {


    public Proceso() {

        T1 tarea1 = new T1();
        Thread t1 = new Thread(tarea1);
        t1.start();

        T2 tarea2 = new T2();
        Thread t2 = new Thread(tarea2);
        t2.start();

    }


}
```

Clase Tarea

```
public abstract class Tarea implements Runnable {
    protected static int y1=0,y2=0;
    protected static int critical=0;

}
```

Dos Hilos

Clase T1

```
public class T1 extends Tarea {

    public T1() {
    }

    public void run() {
        while (true) {
            Tarea.y1 = Tarea.y2 + 1;

            while ((!(Tarea.y2 == 0) && !(Tarea.y1 <= Tarea.y2))) {
                }

            Tarea.critical++;
            Tarea.critical--;
            Tarea.y1 = 0;

//          if (Tarea.critical != 0){
                System.out.println("Valor CRITICAL desde T1 = " + (Tarea.critical));
                System.out.println("Valor Y1 desde T1 = " + (Tarea.y1));
                System.out.println("Valor Y2 desde T1 = " + (Tarea.y2));
//          }
        }
    }

}
```

Dos Hilos

Clase T2

```
public class T2 extends Tarea {

    public T2() {
    }

    public void run() {
        while (true) {
            Tarea.y2 = Tarea.y1 + 1;

            while ((!(Tarea.y1 == 0) && !(Tarea.y2 <= Tarea.y1))) {
            }

            Tarea.critical++;
            Tarea.critical--;

            Tarea.y2 = 0;

//          if (Tarea.critical != 0){
                System.out.println("Valor CRITICAL desde T2 = " + (Tarea.critical));
                System.out.println("Valor Y1 desde T2 = " + (Tarea.y1));
                System.out.println("Valor Y2 desde T2 = " + (Tarea.y2));
//          }

        }
    }

}
```