

Threads – Java Concurrency Cookbook

**Programacion Concurrente
2020**

Ing. Ventre, Luis O.

Creating, Running and setting the characteristics of a Thread

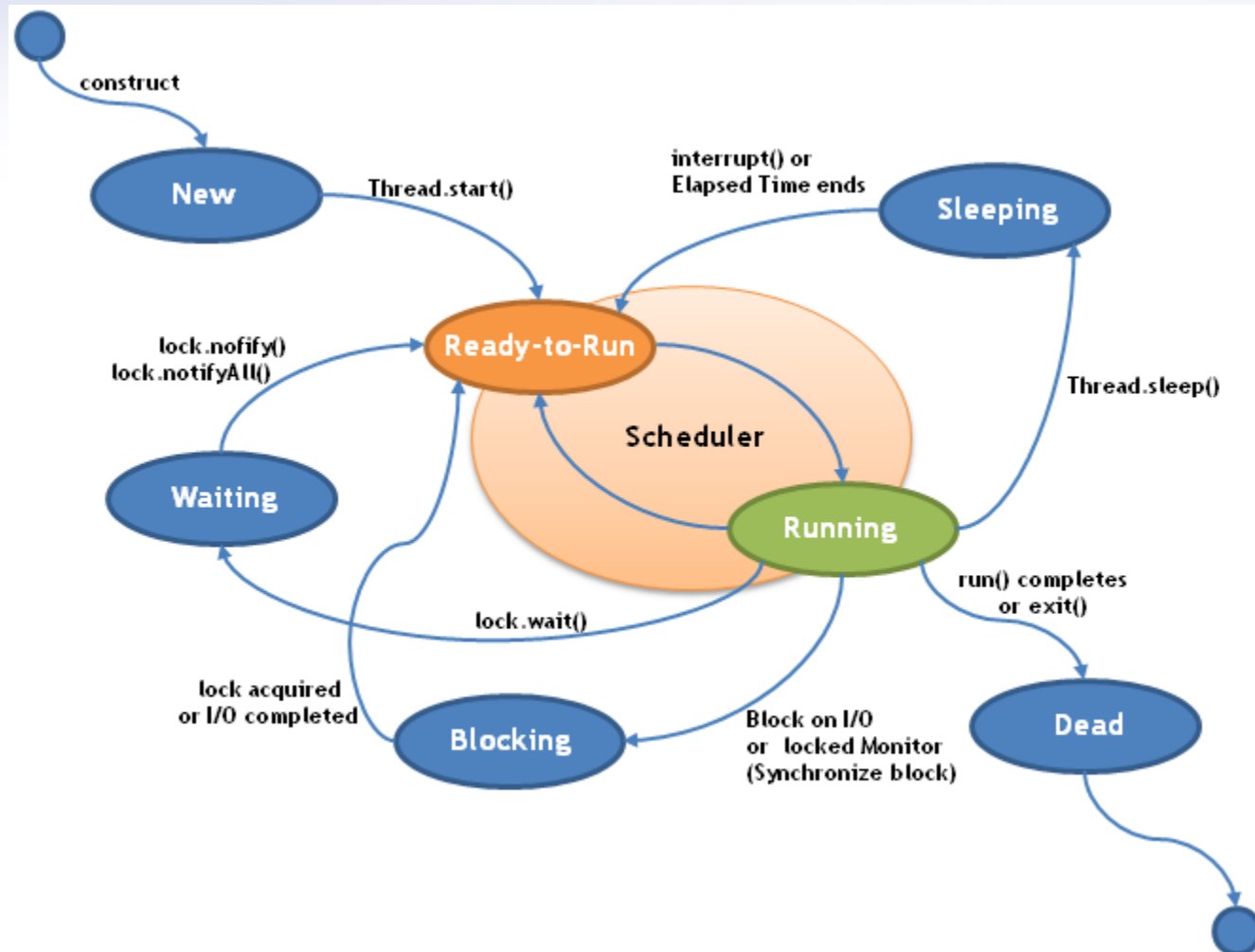
- Aprenderemos como **crear y ejecutar** un Hilo en una aplicación Java y además como **setear** algunos de sus **atributos**.
- Cuando hablamos de concurrencia en el mundo de los procesadores, nos referimos a tareas independientes que se ejecutan simultáneamente en una computadora. Esta simultaneidad puede ser real, si los recursos lo permiten.
- Los hilos (en inglés Threads) son objetos.
 - Hay dos formas de crear Hilos en JAVA:
 - Extendiendo la clase Thread y sobrescribiendo 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; veremos además como cambiar algunos atributos de los hilos.
- Atributos de los objetos de la clase Thread:
 - ID
 - Name
 - Priority (1 a 10)
 - Status

Getting & Setting Thread Info

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



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. Implemente el método run. En éste método, el cual sera ejecutado por los hilos creados se calcularán y contarán números primos hasta un determinado límite.

Creating and Running - HOW

```
@Override
public void run() {
    long current = 1L;
    long max = 200L; //pocos numeros que pasa con prioridad? <200
    long numPrimes = 0L;

    System.out.printf("Thread '%s': START\n", Thread.currentThread().getName());
    while (current <= max) {
        if (isPrime(current)) {
            numPrimes++;
        }
        current++;
    }
    System.out.printf("Thread '%s': END. Number of Primes: %d\n",
        Thread.currentThread().getName(), numPrimes);
}
```

Creating and Running - HOW

3. Ahora se implementará el método auxiliar isPrime()

```
private boolean isPrime(long number) {  
    if (number <= 2) {  
        return true;  
    }  
    for (long i = 2; i < number; i++) {  
        if ((number % i) == 0) {  
            return false;  
        }  
    }  
    return true;  
}
```

Creating and Running - HOW

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

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

5. Se imprime por pantalla info de las prioridades:

```
// Thread priority information  
System.out.printf("Minimum Priority: %s\n", Thread.MIN_PRIORITY);  
System.out.printf("Normal Priority: %s\n", Thread.NORM_PRIORITY);  
System.out.printf("Maximun Priority: %s\n", Thread.MAX_PRIORITY);
```

6. Luego se crean: 10 hilos para ejecutar los 10 objetos tarea; se crean además dos arreglos para almacenar los objetos threads y sus estados. Se setea el atributo prioridad de los hilos pares en máximo y los impares en mínimo.

Creating and Running - HOW

```
Thread threads[];  
Thread.State status[];  
  
// Launch 10 threads to do the operation, 5 with the max  
// priority, 5 with the min  
threads = new Thread[10];  
status = new Thread.State[10];  
for (int i = 0; i < 10; i++) {  
    threads[i] = new Thread(new Calculator());  
    if ((i % 2) == 0) {  
        threads[i].setPriority(Thread.MAX_PRIORITY);  
    } else {  
        threads[i].setPriority(Thread.MIN_PRIORITY);  
    }  
    threads[i].setName("My Thread " + i);  
}
```

Creating and Running - HOW

7. Luego se procede a escribir en un archivo el estado de los hilos antes de lanzar su ejecucion, luego se lanzan los hilos.

```
// Wait for the finalization of the threads. Meanwhile,  
// write the status of those threads in a file  
try (FileWriter file = new FileWriter(".\\data\\log.txt");  
     PrintWriter pw = new PrintWriter(file);) {  
  
    // Write the status of the threads  
    for (int i = 0; i < 10; i++) {  
        pw.println("Main : Status of Thread " + i + " : " + threads[i].getState());  
        status[i] = threads[i].getState();  
    }  
  
    // Start the ten threads  
    for (int i = 0; i < 10; i++) {  
        threads[i].start();  
    }  
}
```

Creating and Running - HOW

8. A continuación se espera la finalización de los hilos:

```
// Wait for the finalization of the threads. We save the status of
// the threads and only write the status if it changes.
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);
    }
}
```

Creating and Running - HOW

9. En el bloque anterior se llamo al metodo writeThreadInfo el cual es:

```
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");  
}
```

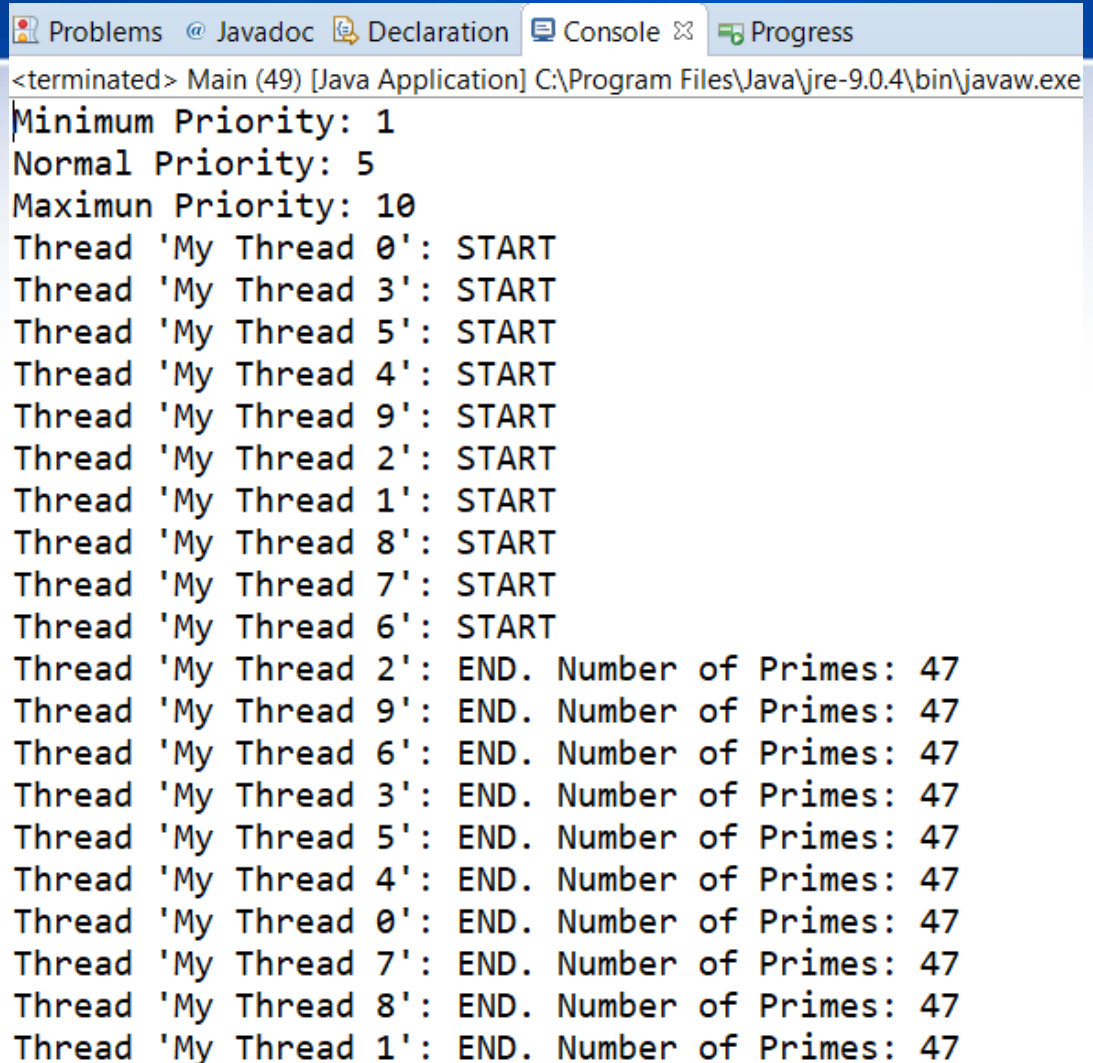
Creating and Running - HOW

- La salida por la consola:

Analice:

-Que sucedió con la prioridad?

-Encuentra algún parámetro a modificar para verificar su funcionamiento?



The screenshot shows a Java IDE console window with the following tabs: Problems, Javadoc, Declaration, Console, and Progress. The console output is as follows:

```
<terminated> Main (49) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe
Minimum Priority: 1
Normal Priority: 5
Maximun Priority: 10
Thread 'My Thread 0': START
Thread 'My Thread 3': START
Thread 'My Thread 5': START
Thread 'My Thread 4': START
Thread 'My Thread 9': START
Thread 'My Thread 2': START
Thread 'My Thread 1': START
Thread 'My Thread 8': START
Thread 'My Thread 7': START
Thread 'My Thread 6': START
Thread 'My Thread 2': END. Number of Primes: 47
Thread 'My Thread 9': END. Number of Primes: 47
Thread 'My Thread 6': END. Number of Primes: 47
Thread 'My Thread 3': END. Number of Primes: 47
Thread 'My Thread 5': END. Number of Primes: 47
Thread 'My Thread 4': END. Number of Primes: 47
Thread 'My Thread 0': END. Number of Primes: 47
Thread 'My Thread 7': END. Number of Primes: 47
Thread 'My Thread 8': END. Number of Primes: 47
Thread 'My Thread 1': END. Number of Primes: 47
```

LOG

- Analizar el documento LOG, modificar valores de límite de números primos a calcular y obtener conclusiones.

```
Main.java Calculator.java log.txt
3Main : Status of Thread 2 : NEW
4Main : Status of Thread 3 : NEW
5Main : Status of Thread 4 : NEW
6Main : Status of Thread 5 : NEW
7Main : Status of Thread 6 : NEW
8Main : Status of Thread 7 : NEW
9Main : Status of Thread 8 : NEW
10Main : Status of Thread 9 : NEW
11Main : Id 12 - My Thread 0
12Main : Priority: 10
13Main : Old State: NEW
14Main : New State: TERMINATED
15Main : *****
16Main : Id 13 - My Thread 1
17Main : Priority: 1
18Main : Old State: NEW
19Main : New State: RUNNABLE
20Main : *****
21Main : Id 14 - My Thread 2
22Main : Priority: 10
23Main : Old State: NEW
24Main : New State: TERMINATED
25Main : *****
26Main : Id 15 - My Thread 3
27Main : Priority: 1
28Main : Old State: NEW
29Main : New State: RUNNABLE
30Main : *****
```

Interrupting A Thread

- Un programa JAVA, con más de un hilo en ejecución solo finaliza, cuando todos sus hilos (no daemons) terminan su ejecución.
- Puede ser necesario finalizar la ejecución de un Hilo. Por ej. Si queremos terminar un programa o finalizar la tarea que el hilo está haciendo.
- Java provee un mecanismo para indicarle al hilo **nuestra voluntad (gentil)** de detenerlo.
- Una peculiaridad del mecanismo, es que el hilo puede ignorar la petición y continuar trabajando.

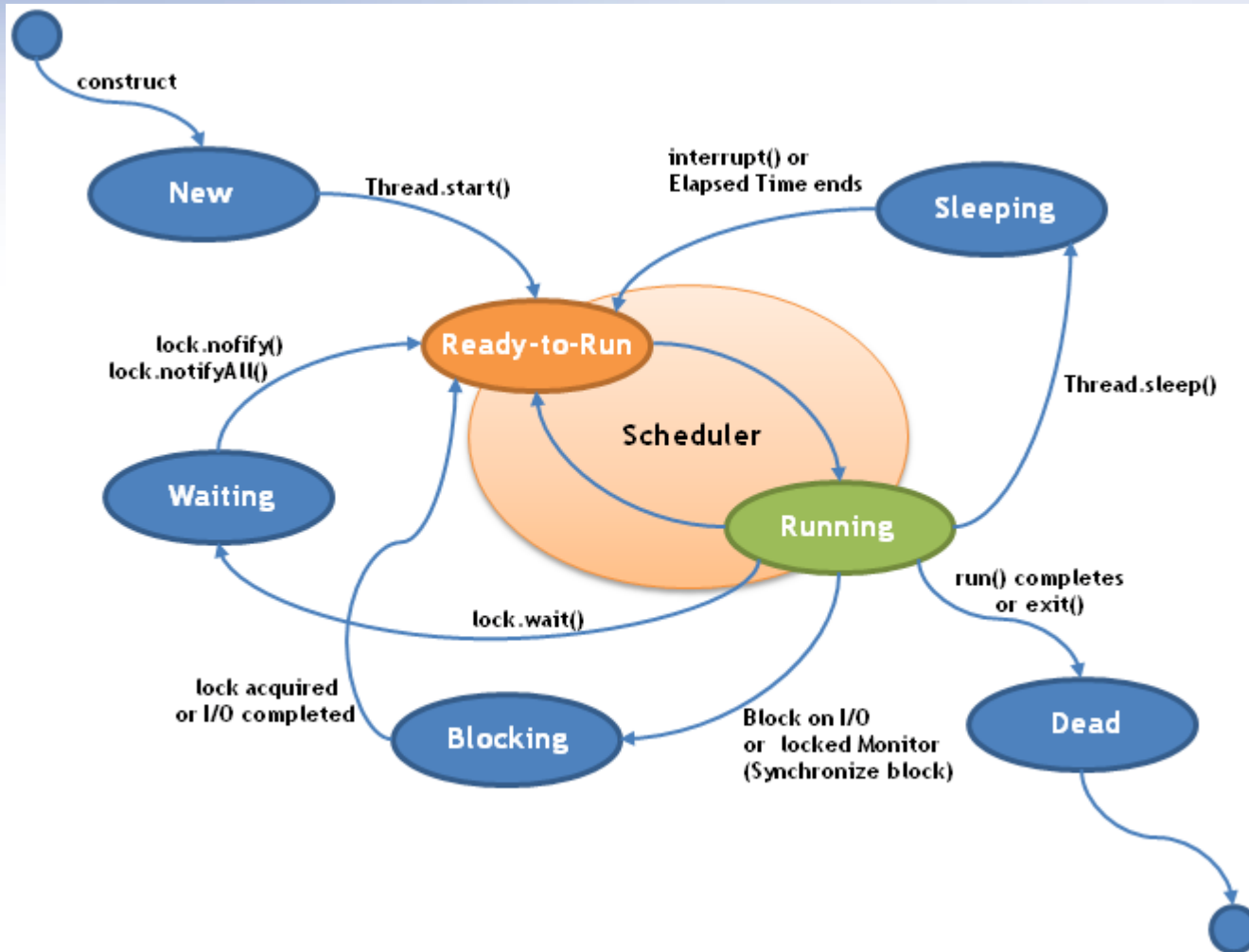
Interrupting A Thread

- Se realizara un programa que crea un hilo y 10 segundos después solicita su fin usando el mecanismo de interrupción.
- Clase Main

```

Main.java  PrimeGenerator.java
1  package Pkt;
2
3  public class Main {
4
5      public static void main(String[] args) {
6          Thread task = new PrimeGenerator();
7          task.start();
8
9          try {
10             Thread.sleep(10000);
11         }
12         catch (InterruptedException e) {
13             e.printStackTrace();
14         }
15
16         task.interrupt();
17     }
18 }
19
```


Interrupting A Thread



Interrupting A Thread

- Cree una clase que extiende la clase Thread.

```
public class PrimeGenerator extends Thread {
```

- Sobrescriba el método run, incluyendo un loop que correrá indefinidamente.
 - En este loop se procesarán números comenzando en 1.
 - Para cada número, se determinara si es primo o no.
 - Si es primo se imprimirá en la consola.
 - Luego de procesar el numero se chequeará si el hilo ha sido interrumpido a través de `isInterrupted()`.
 - El método `isPrime` determinara si es o no primo.

Class PrimeGenerator

```
package Pkt;

public class PrimeGenerator extends Thread {

    @Override
    public void run() {

        long number = 1L;
        while(true){

            if(isPrime(number)){
                System.out.println("Number" + number + " is Prime");
            }

            if (isInterrupted()){
                System.out.printf("The prime generator has been interrupted\n");
                return;
            }
            number++;
        }
    }

    private boolean isPrime(long number) {
        if(number<=2)
            return true;

        for(long i=2;i<number;i++)
        {
            if((number%i)==0)
                return false;
        }
        return true;
    }
}
```

Interrupting A Thread

- Ejecute el programa y observe los resultados.
- La salida por consola debería ser similar a (dependiendo del tiempo de sleep)

```
Number148199 is Prime  
Number148201 is Prime  
Number148207 is Prime  
Number148229 is Prime  
Number148243 is Prime  
Number148249 is Prime  
Number148279 is Prime  
Number148301 is Prime  
Number148303 is Prime  
Number148331 is Prime  
Number148339 is Prime  
The prime generator has been interrupted
```

Interrupting A Thread

- Ver en la IDE el ejemplo de la version 2 del cookbook. Tiene impresión de flags de estado del hilo.

Console

<terminated> Main (52) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (26 mar. 2019 18:32:39)

Number 111227 is Prime

Number 111229 is Prime

Number 111253 is Prime

Number 111263 is Prime

Number 111269 is Prime

Number 111271 is Prime

Main: Status of the Thread: RUNNABLE

Main: isInterrupted: true

Main: isAlive: true

Number 111301 is Prime

The Prime Generator has been Interrupted

Interrupting A Thread

- Que sucede si se coloca un sleep, luego del interrupt en el main, antes de imprimir los valores de los flags?

Console

<terminated> Main (52) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (26 mar. 2019 18:33:30)

```
Number 111977 is Prime
Number 111997 is Prime
Number 112019 is Prime
Number 112031 is Prime
Number 112061 is Prime
Number 112067 is Prime
Number 112069 is Prime
The Prime Generator has been Interrupted
Main: Status of the Thread: TERMINATED
Main: isInterrupted: false
Main: isAlive: false
```

Interrupting A Thread

- El mecanismo visto para la interrupción de un hilo es adecuado para programas simples. Sin recursividad ni complejidad en cantidad de métodos.
- Java provee la interrupción por excepción `InterruptedException`, esta interrupción puede ser lanzada al detectar la interrupción del hilo y hacer el catch desde el método `run()`.

Interrupting A Thread

- El siguiente ejercicio implementará un hilo que busca archivos con un determinado nombre en un directorio y sus subdirectorios.
- Crear una clase llamada FileSearch que implemente la interfaz runnable.
- Declarar dos atributos privados, uno para el nombre del archivo y el otro para el directorio donde comenzar a buscar.

Interrupting A Thread

- Implemente el método `run()`, la cual chequea si el atributo `fileName` es un directorio, y si lo es llama al método `directoryProcess()`.
 - Durante la ejecución de este método, puede ocurrir una excepción por Interrupción (`InterruptedException`) por lo que debemos implementar `catch` correspondiente.
- Implementar el método `directoryProcess()`. En cada llamado al método hará una recursión llamando al método `fileProcess()`.

Interrupting A Thread

- Luego de procesar todos los archivos y directorios el hilo chequea si ha sido interrumpido y lanza la excepción correspondiente.
- Implementar el método `fileProcess`, este método compara el archivo buscado con cada uno, si son iguales, imprime un mensaje en la consola. Luego el hilo chequea si ha sido interrumpido y lanza la excepción correspondiente.

Interrupting A Thread

- Implemente la clase Main.
- Cree e inicialice un objeto de la clase FileSearch.
- Comience la ejecución del hilo.
- Espere 10 segundos
- Interrumpa el hilo.
- Ejecute y vea el resultado.

Interrupting A Thread

- Clase Main

```
*Main.java  FileSearch.java
1  package Pkt;
2
3  import java.util.concurrent.TimeUnit;
4
5  public class Main {
6
7      public static void main(String[] args) {
8
9          FileSearch buscador = new FileSearch("C:\\", "log.txt");
10         Thread hilo = new Thread(buscador);
11
12         hilo.start(); // si pongo run, el hilo nunca se detiene porque?
13
14         try{
15             TimeUnit.SECONDS.sleep(10);
16         } catch (InterruptedException e){
17             e.printStackTrace();
18         }
19
20         hilo.interrupt();
21     }
22
23 }
```

Interrupting A Thread

- Clase FileSearch

```
import java.io.File;

public class FileSearch implements Runnable {

    private String initPath;
    private String fileName;

    public FileSearch(String initPath, String fileName) {
        super();
        this.initPath = initPath;
        this.fileName = fileName;
    }

    public void run() {
        File file=new File(initPath);
        if(file.isDirectory()){
            try{
                directoryProcess(file);
            }catch (InterruptedException e){
                System.out.printf("%s: La busqueda ha sido interrumpida ",
                                Thread.currentThread().getName());
            }
        }
    }
}
```

Interrupting A Thread

- Método directoryProcess

```
private void directoryProcess(File file) throws InterruptedException {  
  
    File list[]=file.listFiles();  
    if(list != null){  
        for(int i=0;i<list.length;i++){  
            if(list[i].isDirectory()){  
                directoryProcess(list[i]);  
            }else{  
                fileProcess(list[i]);  
            }  
        }  
    }  
  
    if(Thread.interrupted()){  
        throw new InterruptedException();  
    }  
}
```

Interrupting A Thread

- Método fileProcess

```
private void fileProcess(File file) throws InterruptedException {  
    if (file.getName().equals(fileName)) {  
        System.out.printf("%s : %s\n", Thread.currentThread().getName(),  
                           file.getAbsolutePath());  
    }  
  
    if(Thread.interrupted())  
    {throw new InterruptedException();  
    }  
}
```

Interrupting A Thread

- Argumente porque al modificar el código y ejecutar la instrucción `run` desde el `main` en lugar de `start`, el buscador SI encuentra el archivo buscado.
- Y al ejecutar la instrucción `start()` no la encuentra.
- Cual puede ser el posible problema?
- Alguna solución?

Sleeping & Resuming a Thread

- En diferentes ocasiones es necesario **suspender la ejecución de un hilo por un determinado tiempo.**
- Por ej. un hilo que lee un sensor, y luego durante un minuto no realiza tarea alguna.
- Puede utilizar el metodo `sleep()` de la clase `Thread` para este propósito.

Sleeping & Resuming a Thread

- Este método recibe un entero como argumento, que representa el número en **milisegundos** que el hilo suspende su ejecución.
- Cuando el tiempo de sleep finaliza, el hilo continúa con la ejecución de la instrucción siguiente al sleep(). Cuando la JVM le asigne cpu time.
- *Otra posibilidad es utilizar el método sleep() del enumerado TimeUnit.*

Sleeping & Resuming a Thread

- El siguiente programa utiliza el método sleep() para mostrar por la consola la fecha a cada segundo.
- Paso a paso:
- 1-Crear la clase llamada FileClock y especificar que implementa la interfaz runnable

```
public class FileClock implements Runnable {
```

Sleeping & Resuming a Thread

- 2-Implemente el método run()
 - Cree un bucle con 10 iteraciones. En cada iteración cree un objeto Date.
 - Luego llame al método sleep() de la clase TimeUnit, atributo SECONDS y suspenda la ejecución del hilo por un segundo.
 - Como el método sleep **puede arrojar una InterruptedException**, se debe incluir el código para hacer el catch correspondiente.

Sleeping & Resuming a Thread

- 2-Implemente el método run()

```
public void run() {  
    for(int i=0;i<10;i++){  
        System.out.printf("%s\n", new Date());  
        try{  
            TimeUnit.SECONDS.sleep(1);  
        }catch (InterruptedException e){  
            System.out.printf("The FileClock has been interrupted");  
            return;  
        }  
    }  
}
```

Sleeping & Resuming a Thread

- 3-Hemos creado el hilo. Ahora crearemos la clase main.
 - Crear un objeto de la clase FileClock.
 - Luego un Thread para ejecutarlo. Y start()

```
public class FileMain {  
  
    public static void main(String[] args) {  
  
        FileClock clock= new FileClock();  
        Thread thread=new Thread(clock);  
  
        thread.start();  
    }  
}
```

Sleeping & Resuming a Thread

- 4-Llamar al método sleep() por 5 segundos, desde la clase main.
 - Luego interrumpir el hilo.
 - Ejecute y vea los resultados!.

```
try{  
    TimeUnit.SECONDS.sleep(5);  
}catch (InterruptedException e){  
    e.printStackTrace();  
}
```

```
thread.interrupt();  
}
```

Sleeping & Resuming a Thread

- En el ejercicio anterior, al ejecutar `interrupt()` el hilo finalizaba.
- En este ejercicio al ejecutar `interrupt()` que sucede?
- Porque?
- Comprende la gentileza y la diferencia con el metodo deprecado `thread.stop()`.

Waiting for the finalization

- En algunas situaciones, debemos esperar a que un hilo 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 del `Thread` llamado.

Waiting for the finalization

- Ejemplo...
- Supongamos que en una clase hay diferentes grupos.
- Nuestro programa debe crear un hilo para administrar la impresion de mensajes de cada grupo
- Desde main se crean los hilos, y se asignan los grupos como objetos que implementan la interfaz runnable.

Waiting for the finalization

- Es deseado que los mensajes se impriman de manera ordenada.
- Y el main debe esperar que cada grupo haya impreso sus mensajes antes de comenzar con el siguiente.

Waiting for the finalization

- Main

Main.java Grupo.java

```
3 public class Main {
4
5     public static void main(String[] args) throws InterruptedException {
6         // TODO Auto-generated method stub
7
8         Thread teapots=new Thread(new Grupo("Teapots"));
9         Thread venThread=new Thread(new Grupo("VenThread"));
10        Thread giA=new Thread(new Grupo("GiA"));
11
12        teapots.start();
13        teapots.join();
14
15        venThread.start();
16        venThread.join();
17
18        giA.start();
19        giA.join();
20
21        //teapots.join();
22        //venThread.join();
23        //giA.join();
24
25
26        System.out.printf("Main: Se finalizo la impresion de mensajes %s\n", new Date());
```

Waiting for the finalization

- Clase Grupo

```
Main.java  Grupo.java ✕
1
2 public class Grupo implements Runnable {
3
4     String mensaje;
5
6     public Grupo(String nombre){
7
8         mensaje = "Hola, somos el grupo " + nombre + " y este es nuestro mensaje numero: ";
9
10    }
11    @Override
12    public void run(){
13        // TODO Auto-generated method stub
14
15        for (int i=1; i<100;i++){
16            String msj = mensaje + i;
17            System.out.println(msj);
18        }
19    }
20
21 }
22
23 }
24
```

Waiting for the finalization

- Que sucederia si comentamos el primer join?.
- Que sucederia si comentamos todos los join, y los colocamos luego de todos los starts?
- Que sucederia si no ponemos joins?

Waiting for the finalization

- Veremos otro 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) {
        DataSourceLoader dsLoader = new DataSourceLoader();
        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 finalice
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

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

Waiting for the finalization

- La classe DataSourceLoader

```
⊕ import java.util.Date;

public class DataSourceLoader 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 classe 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:

 Console 

```
<terminated> Main (3) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (4 de abr. de 2017 4:51:02 p. m.)  
Beginning data sources loading: Tue Apr 04 16:51:02 ART 2017  
Beginning Network Connections sources loading: Tue Apr 04 16:51:02 ART 2017  
Data sources loading has finished: Tue Apr 04 16:51:06 ART 2017  
Network Connections sources loading has finished: Tue Apr 04 16:51:11 ART 2017  
Main: Configuration has been loaded: Tue Apr 04 16:51:11 ART 2017
```

HOMework

1. Es necesario implementar todos los proyectos vistos en esta clase, en la ide de su preferencia.
-De esta forma debe: entender los conceptos y aprenderlos.
2. Definir los grupos, una vez definidos enviar NOMBRE identificador del grupo (4 integrantes, excepción 5)
3. Comenzar con diagrama de clases y secuencia.
4. Analizar el siguiente ejercicio para conversar sobre el mismo y sacar conclusiones prox clase.

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...

Ejemplo

```
int y1 = 0;  
int y2 = 0;  
short in_critical = 0;
```

```
active proctype process_1() {  
  do  
    :: true ->  
      y1 = y2+1;  
      ((y2==0) || (y1<=y2));  
      in_critical++;  
      in_critical--;  
      y1 = 0;  
  od  
}
```

```
active proctype process_2() {  
  do  
    :: true ->  
      y2 = y1+1;  
      ((y1==0) || (y2<y1));  
      in_critical++;  
      in_critical--;  
      y2 = 0;  
  od  
}
```

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));  
            //      }  
        }  
    }  
}
```

Modifique el código de la clase T1

Clase T1:

Luego de la instrucción

`Tarea.y1 = 0;`

Coloque una instrucción de SLEEP por 0 milisegundos.

try {

`Thread.sleep(0);`

} catch (InterruptedException e) {

// TODO Auto-generated catch block

`e.printStackTrace();`

}

Modifique el código de la clase T1

Intente ejecutar el código y lograr que se imprima en pantalla algún valor de CRITICAL!

Modifique el código de la clase T1

Clase T2:

Luego de la instrucción

Tarea.critical++;

Coloque una instrucción de SLEEP por 0 milisegundos.

try {

Thread.sleep(0);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

Puede explicar la siguiente ejecución

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Displays the project structure. The package `Ejercicio1Clase2-DosHilosSeccionCritica` contains a `src` folder with `Main.java` and `Tareas`. The `Tareas` package contains `Proceso.java`, `T1.java`, `T2.java`, and `Tarea.java`.
- Editor:** Shows the source code of `T1.java`. The code defines a `T1` class that extends `Tarea`. It has a `run()` method that enters a `while` loop. Inside the loop, it increments `Tarea.y1` and `Tarea.cantidad`. There is a `while` loop that checks for a condition involving `Tarea.y2` and `Tarea.y1`. Inside this loop, there is a `try` block that calls `Thread.sleep(0);` and a `catch` block for `InterruptedException`. After the `while` loop, it increments `Tarea.critical` and decrements `Tarea.critical--`.
- Console:** Shows the output of the program. The output indicates that the program terminated and shows the values of `CRITICAL`, `Y1`, and `Y2` for different iterations of the loop. The output is as follows:

```
<terminated> Main (29) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (14 mar. 2019 15:17:07)
-----Valor CRITICAL desde T1 = -1
-----Valor Y1 desde T1 = 0
-----Valor Y2 desde T1 = 0
Cantidad123498
-----Valor CRITICAL desde T1 = -1
-----Valor Y1 desde T1 = 0
-----Valor Y2 desde T1 = 0
Cantidad123499
-----Valor CRITICAL desde T1 = -1
-----Valor Y1 desde T1 = 0
-----Valor Y2 desde T1 = 0
Cantidad123500
-----Valor CRITICAL desde T1 = -1
```

Puede explicar la siguiente ejecución

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Displays the project structure. The active project is 'Ejercicio1Clase2-DosHilosSeccionCritica', which contains a 'src' package with files 'Main.java', 'T1.java', 'T2.java', and 'Tarea.java'.
- Editor:** Shows the code for 'T2.java'. The code defines a class 'T2' that extends 'Tarea'. It has a 'run()' method containing a 'while' loop that increments 'Tarea.y2' and 'Tarea.cantidad' while 'Tarea.y1' is not zero. A 'try-catch' block handles 'InterruptedException'.
- Console:** Shows the output of the program. It displays the execution of 'Main (29)' and the output of 'T2.run()' for two threads, T1 and T2. The output shows that both threads increment 'cantidad' to 491503, indicating a race condition where both threads executed the critical section simultaneously.

```
1 package tareas;
2
3 public class T2 extends Tarea {
4
5     public T2() {
6     }
7
8     public void run() {
9         while (true) {
10             Tarea.y2 = Tarea.y1 + 1;
11             Tarea.cantidad +=1;
12
13             while ((!(Tarea.y1 == 0) && !(Tarea.y2 <= Tarea.y1))) {
14                 //System.out.println("-----Trabado desde T2 = ");
15                 try {
16                     Thread.sleep(0);
17                 } catch (InterruptedException e) {
18                     // TODO Auto-generated catch block
19                     e.printStackTrace();
20                 }
21             }
22
23             Tarea.critical++;
24
25             try {
```

Console Output:

```
Main (29) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (14 mar. 2019 15:24:30)
-----Valor CRITICAL desde T1 = -3
-----Valor Y1 desde T1 = 0
-----Valor Y2 desde T1 = 0
Cantidad491503
-----Valor CRITICAL desde T1 = -3
-----Valor Y1 desde T1 = 0
-----Valor Y2 desde T1 = 0
Cantidad491504
-----Valor CRITICAL desde T1 = -3
-----Valor Y1 desde T1 = 0
-----Valor Y2 desde T1 = 0
Cantidad491505
```


THREAD SAFE

Característica que brinda la posibilidad de ejecutar el código en un entorno con múltiples hilos garantizando consistencia?

Los atributos estáticos son THREAD SAFE?

En base a lo visto que respondería?.

Busque la respuesta y analice lo sucedido.

Que debemos hacer para hacer nuestro código

THREAD SAFE