Hilos (procesos ligeros)

Dentro de un programa, un hilo es un flujo de control. Los hilos múltiples y los procesos múltiples son diferentes. La diferencia clave es que los hilos múltiples en un programa comparten el mismo estado y se ejecutan en el mismo espacio de direcciones.

Un programa de java puede tener mas de un hilo de ejecución. Los hilos son derivados de la clase Thread. Todos los hilos tienen atributos básicos: un cuerpo, un estado y una prioridad.

El cuerpo de un hilo

El cuerpo de un hilo es su núcleo: el contenido que está dentro del método run(). Esta sección de código define la acción de un hilo.

Estado

Durante su existencia un hilo puede estar en diferentes estados . El estado indica lo que el hilo esta haciendo y lo que pude hacer.

Prioridad

Los hilos tienen una prioridad predefinida y predeterminada que heredan del hilo padre. Se puede cambiar la prioridad de un hilo en cualquier momento usando el metodo setPriority(). Los niveles de prioridad se asignan entre 1 y 10 (el uno el menor el 10 el mayor).

Cuando la calendarización de Java necesita elegir un hilo para ejecutar, seleccionara el hilo ejecutable con el nivel de prioridad más alta.

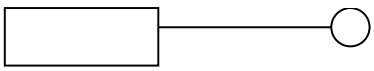
Ejemplo de un programa multihilo

```
//Printer.java
public class Printer implements Runnable {
        Thread printerThread;
        String string;
        int count;
        int sleepTime;

        public Printer(String s, int howMany, int sleep){
                count=howMany;
                string=s;
                sleepTime=sleep;
                printerThread=new Thread(this);
                printerThread.start();
        }

        public void run(){
                while(count-- >0){
                        System.out.println(string);
```

```
                        try{
                                Thread.sleep(sleepTime);
                        } catch (Exception e) {
                                return;
                        }
                }
        }

        public static void main(String args[]){
                new Printer("Ping", 5, 300);
                new Printer("Pong", 5, 500);
        }
}
```

El programa puede producir la siguiente salida

```
Ping
Pong
Ping
Pong
Ping
Ping
Pong
Ping
Pong
Pong
```



Notación UML para el código Ejemplo

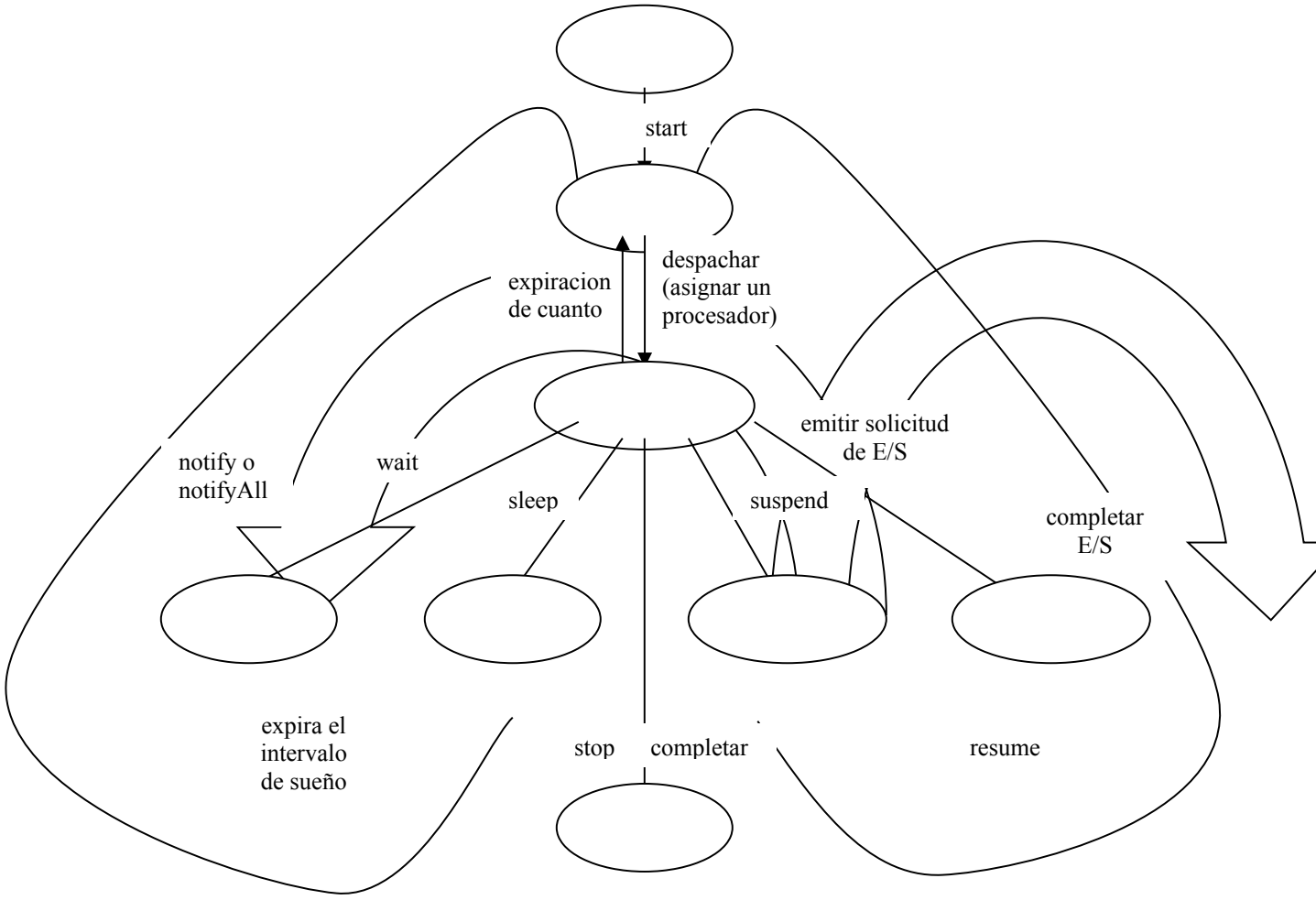
La interface Runnable se encuentra en el paquete java.lang y se define básicamente como:

```
package java.lang;

public interface Runnable {
        void run();
}
```

Un clase implementa la interfaz Runnable cuando no le es posible heredar de la clase Thread. Como se puede observar del ejemplo de arriba si se implementa esta interfaz se declara una variable de tipo Thread y se crea un objeto de dicho tipo. Obsérvese además que al constructor de la clase Thread se le pasa como argumento una instancia de un objeto que implemente la interfaz Runnable.

Ciclo de vida de los hilos

Todos los hilos, después de su creación y antes de proceder a su eliminación, pasarán por uno de estos ocho estados: nacido, listo, en ejecución, en espera, dormido, suspendido, bloqueado o muerto.



Ciclo de vida de un hilo

Esqueleto para un applet para realizar animación

El código siguiente muestra como se pueden usar los hilos para que un applet muestre una animación. En dicho código observe como las llamadas a los métodos de la clase Thread hace pasar al hilo por varios estados de los mostrados en el diagrama anterior.

```
import java.awt.*;
import java.applet.*;
```

```
public class AppletAnimador extends Applet implements Runnable {
        Thread animador;
      public void init(){
        animador=new Thread(this);
      }
      public void start(){
        if(animador.isAlive()){
                animador.resume();
        }
        else
                animador.start();
      }
      public void stop(){
        animador.suspend();
      }
      public void destroy(){
        animador.stop();
      }
      public void run(){
        while(true){
                repaint();
                Thread.sleep(500);//dormir por un tiempo
                //avanzar al siguiente cuadro
        }
      }
      public void paint(Graphics g){
        //pinta el cuadro actual
      }
}
```

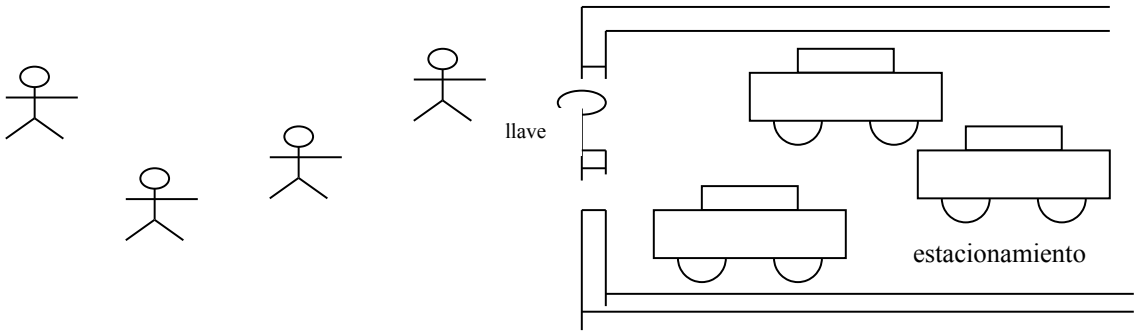
Monitores

Definición de monitor de sistemas operativos

Un monitor es una colección de procedimientos, variables y estructuras de datos que se agrupan en cierto tipo particular de modulo o paquete. Los monitores sirven para conseguir la exclusión mutua por que solo un proceso puede estar activo en un monitor en cada momento.

Definición de monitor de Java

Los monitores son un esquema de sincronización que usa java para sincronizar el acceso a datos compartidos. En este esquema cada objeto tiene un objeto llamado monitor asociado con el. Este monitor actúa como un portero para el objeto y solo permitirá un método sincronizado a la vez. Cuando el método sincronizado finaliza, el monitor se desbloquea, dándole a otro método sincronizado la oportunidad de empezar a ejecutarse.



Los hilos necesitan una misma llave para acceder a los recursos que protege un monitor

Dado que el orden en que se ejecutan los hilos no es predecible, si se permite que varios hilos modifiquen los datos simultáneamente, los datos podrían quedar en un estado inconsistente. Por ejemplo suponga que un objeto tiene dos campos que necesitan mantenerse consistentes: uno es un arreglo de números y el otro es una cuenta de cuantos números hay en el arreglo.

Un programa que puede resultar en datos inconsistentes

```
class NumberArray {
        int count;
        int numbers[]=new int[20];
        void addNumber(int n){
                numbers[count]=n;
                count++;
        }
        void print(){
                for(int i=0; i < count; i++){
                        System.out.println(numbers[i]);
                }
        }
}
```

Suponga que dos hilos ejecutaran el metodo addNumber. Y la siguiente secuencia que puede conducir a un estado inconsistente:

```
Hilo1 : numbers[count]=12;
Hilo2:  numbers[count]=13;
Hilo1:  count++;
Hilo2:  count++;
```

Un programa con datos consistentes usando métodos sincronizados

```
class NumberArray {
```

```
        int count;
        int number[]=new numbers[20];

        synchronized void addNumber(int n){
                numbers[count]=n;
                count++;
        }
        synchronized void print(){
                for(int i=0; i < count; i++){
                        System.out.println(numbers[i]);
                }
        }
}
```

Relación productor/consumidor con sincronización de hilos

Un hilo ejecutándose en un método synchronized podría determinar que no puede continuar, así que el hilo invoca wait voluntariamente. Esto hace que el hilo deje de competir por el procesador y por el objeto monitor. Ahora, el hilo esperara en una cola mientras otros hilos tratan de entrar en el objeto. Cuando un hilo que está ejecutando un método synchronized termina el objeto puede notificar (con notify) a un hilo en espera que vuelva a estar listo para que pueda intentar obtener otra vez el candado del objeto monitor y ejecutarse. El notify actúa como una señal para el hilo en espera, indicándole que la condición que estaba esperando ya se satisfizo, y que es aceptable que vuelva a entrar en el monitor. Si un hilo invoca notifyAll, todos los hilos que estaban esperando el objeto se hacen elegibles para volver a entrar en el monitor (es decir, todos pasan al estado listo).

```
// SharedCell.java
// Muestra multiples hilos modificando un objeto compartido. Usa sincronización para
//asegurar que ambos hilos accedan a la celda compartida correctamente.

public class SharedCell {
  public static void main( String args[] )
  {
    HoldInteger h = new HoldInteger();
    ProduceInteger p = new ProduceInteger( h );
    ConsumeInteger c = new ConsumeInteger( h );
    p.start();
    c.start();
  }
}



class ProduceInteger extends Thread {
  private HoldInteger pHold;
```

```
  public ProduceInteger( HoldInteger h )
  {
    pHold = h;
  }

  public void run()
  {
    for ( int count = 0; count < 10; count++ ) {
      // sleep for a random interval
      try {
        Thread.sleep( (int) ( Math.random() * 3000 ) );
      }
      catch( InterruptedException e ) {
        System.err.println( e.toString() );
      }

      pHold.setSharedInt( count );
      System.out.println( "Producer set sharedInt to " +
                          count );
    }

    pHold.setMoreData( false );
  }
}

class ConsumeInteger extends Thread {
  private HoldInteger cHold;

  public ConsumeInteger( HoldInteger h )
  {
    cHold = h;
  }
  public void run()
  {
    int val;

    while ( cHold.hasMoreData() ) {
      // sleep for a random interval
      try {
        Thread.sleep( (int) ( Math.random() * 3000 ) );
      }
      catch( InterruptedException e ) {
        System.err.println( e.toString() );
      }
      val = cHold.getSharedInt();
      System.out.println( "Consumer retrieved " + val );
    }
```

```
  }
}

class HoldInteger {
  private int sharedInt = -1;
  private boolean moreData = true;
  private boolean writeable = true;

  public synchronized void setSharedInt( int val )
  {
    while ( !writeable ) {
      try {
        wait();
      }
      catch ( InterruptedException e ) {
        System.err.println( "Exception: " + e.toString() );
      }
    }
    sharedInt = val;
    writeable = false;
    notify();
  }

public synchronized int getSharedInt()
  {
    while ( writeable ) {
      try {
        wait();
      }
      catch ( InterruptedException e ) {
        System.err.println( "Exception: " + e.toString() );
      }
    }
    writeable = true;
    notify();
    return sharedInt;
  }
  public void setMoreData( boolean b ) { moreData = b; }

  public boolean hasMoreData() { return moreData; }
}
```

Programación en RED

La información se mueve entre dos maquinas en forma de paquetes. Donde un paquete contiene como mínimo los tres componentes siguientes: dirección de origen, dirección de destino y datos.

Origen	Destino	Datos

Uno de protocolos mas usados en internet es el TCP/IP (Transmision Control Protocol/Internet Protocol).

Niveles de internet		Niveles OSI
TCP	UDP	Transporte
IP		Red

Arquitectura de TCP/IP y relación con los niveles OSI.

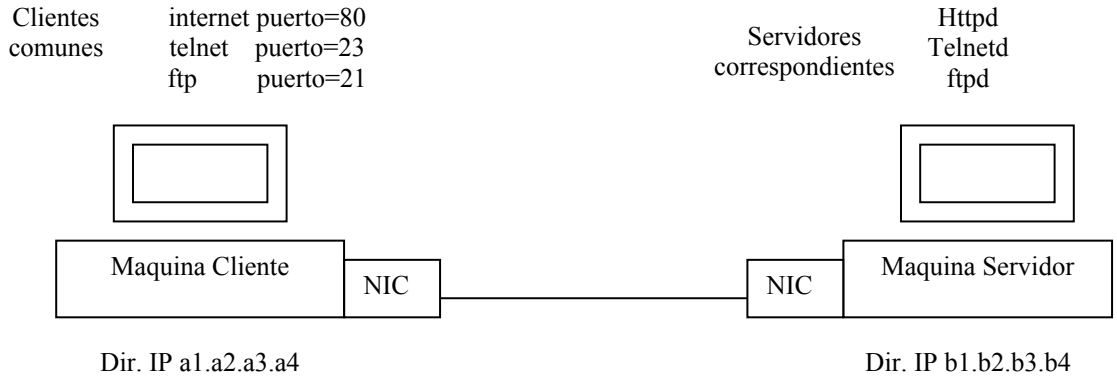
Las direcciones del protocolo IP son de la forma n1.n2.n3.n4 (donde cada ni es un valor de 0 a 225).

Modelo Cliente-Servidor
El modelo cliente-servidor es el modelo estándar de ejecución de aplicaciones en una red.

Servidor
Un servidor es un proceso que se está ejecutando en un nodo de la red y que gestiona el acceso a un determinado recurso.

Cliente

Un cliente es un proceso que se ejecuta en el mismo nodo o en un nodo diferente y que realiza peticiones de servicio al servidor. Las peticiones están originadas por la necesidad de acceder al recurso que gestiona el servidor.



Modelo cliente-servidor en internet