

CLASE 4 – 2.024

MODELO DE DESARROLLO DE PROGRAMAS Y PROGRAMACION CONCURRENTES

FAC.DE INGENIERIA - UNJu

EXCEPCIONES – WAIT - NOTIFY

EXCEPCIONES CON TRY - CATCH



En programación **siempre se producen errores q son necesarios gestionar y tratar correctamente.**



```
Procedure Novel (Book)
Var
  Author : Text;
  Title : Text;
begin
  Fiction:=True; //ye
Try
  if Fiction then //
    exit; // hama.t
  [-]

Catch
  begin (this is th
    end;
  [-]

Finally
  // ah screw it, ju
  Println("Invalid
  // no one reads :
  [-]
```

Java dispone de un mecanismo consistente en el uso de bloques try/catch/finally. La técnica consiste en colocar instrucciones q podrían provocar problemas dentro de 1 bloque try, y colocar a continuación 1 o + bloques catch, de tal forma q si se provoca 1 error de 1 determinado tipo, se salta el bloque catch capaz de gestionar ese tipo de error específico.

El bloque catch contiene el código necesario p/gestionar el tipo específico d error. **SI NO HAY ERRORES** en el bloque try, nunca se ejecutan los bloques catch.

EXCEPCIONES CON TRY - CATCH

```
try
{ //Código que puede provocar errores }
catch(Tipo1 var1) { //Gestión del error var1, de tipo Tipo1 }
...
[ catch(TipoN varN) { //Gestión del error varN, de tipo TipoN } ]
[ finally { //Código de finally } ]
```

```
Procedure Novel (Book)
Var
  Author : Text;
  Title : Text;
begin
  Fiction:=True; // yes
Try
  if Fiction then //
    exit; // hamma
  [-]
Catch
  begin (this is the
    end;
  [-]
Finally
  // ah screw it, ju
  Println("Invalid
  // no one reads
  [-]
```

Es obligat.q exista la zona try, donde se colocan las instrucc. con problemas. Después vienen 1 o + zonas catch, c/u especializada en 1 tipo de error o excepción.

X último está la zona **finally**, encargada de tener 1 código q se ejecuta siempre, independientemente de si se produjeron o no errores.

C/catch se parece a 1 función en la cual sólo se recibe 1 objeto de 1 determinado tipo (el tipo del error). Es decir solo se llama al catch cuyo argumento sea coincidente en tipo con el tipo del error generado.

EXCEPCIONES CON TRY – CATCH (EJEMPLO)

```
public class Try1
{ public static void main(String arg[])
{
    int [] array = new int[20];
    array[-3] = 24;
}
}
```



```
Procedure Novel (Book)
Var
    Author : Text;
    Title : Text;
begin
    Fiction:=True; // yes
Try
    if Fiction then
        exit; // hanna
    [-]

Catch
    begin (this is the
    end;
    [-]

Finally
    // ah screw it, just
    Println("Invalid
    // no one reads
    [-]
```

Al ejecutar se genera el siguiente error:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException at Try1.main(Try1.java:6)



Indica q se ha generado 1 excepción del tipo java.lang.ArrayIndexOutOfBoundsException en la función Try1.main, dentro del fichero Try1.java y en la línea 6 del código. Esta excepción en particular se lanza cuando se intenta acceder a 1 posición de un array y no existe dicha posición. Ahora se procede a gestionar esta interrupción mediante un bloque try/catch, el fichero a crear es Try2.java

EXCEPCIONES CON TRY – CATCH (EJEMPLO)

```
public class Try2
```

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

```
{ int [] array = new int[20];
```

```
try { array[-3] = 24; }
```

```
catch(ArrayIndexOutOfBoundsException excepcion)
```

```
{ System.out.println(" Error de índice en un array"); } }
```



```
public class Try3
```

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

```
{ int [] array = new int[20];
```

```
try { int b = 0; int a = 23/b; }
```

```
catch(ArrayIndexOutOfBoundsException excepcion)
```

```
{ System.out.println(" Error de índice en un array"); }
```

```
catch(ArithmeticException excepcion)
```

```
{ System.out.println(" Error Aritmético"); } }
```



```
Procedure Name: try
Var:
Author: Text:
Title: Text:
begin
  Fiction:=True; // yes
  Try
    if Fiction then //
      exit; // hanna t
    [-]
  Catch
    begin (this is the
      end;
      [-]
  Finally
    // so screw it, j
    printn("invalid
    // no one reads
    [-]
```

Se provoca
1 error de
tipo división
x cero y se
pone un
catch
específico
p/dicho
error
(fichero
Try3.java):

CONCURRENCIA EN JAVA CON WAIT() Y NOTIFY()



A veces conviene q 1 hilo s bloquee a la espera d q ocurra algún evento, como la llegada de 1 dato p/tratar q el Us. termine d escribir algo en 1 interface d Us. **Todos los objetos java tienen el mét.wait() q deja bloqueado al hilo q lo llama y el mét.notify(), q desbloquea a los hilos bloqueados por wait() (modelo productor/consumidor).**

BLOQUEAR UN HILO

Las llamadas a wait() lanzan excepciones q hay q capturar, x lo q todas las llamadas deben estar en 1 bloque try-catch (x simplicidad esto no se hará).

P/q 1 hilo se bloquee basta con q llame al mét.wait() d cualquier objeto. Es necesario q dicho hilo haya marcado ese objeto como ocupado con synchronized. Si no se hace así, salta 1 excepción de q "el hilo no es propietario del monitor" o algo similar.

CONCURRENCIA EN JAVA CON WAIT() Y NOTIFY()

Wait & Notify
in Java



Si el hilo quiere retirar datos de 1 lista y no hay datos, puede esperar a q los haya:

```
synchronized(lista);  
{  
    if (lista.size()==0)  
        lista.wait();  
    dato = lista.get(0);  
    lista.remove(0);  
}
```

Se usa `synchronized(lista)` p/"apropiarse" del objeto lista. Si no hay datos se ejecuta `lista.wait()`. 1 vez q realiza el `wait()`, el obj.lista s marca como "desocupado", d forma q otros hilos lo usen. Cuando se despierta y sale del `wait()` s marca como "ocupado."

El hilo se desbloquea y sale del `wait()` cuando alguien llame a `lista.notify()`. Si el hilo q mete datos en la lista llama luego a `lista.notify()`, cuando sale del `wait()` s tienen datos disponibles en la lista, así q únicamente hay q leerlos (y borrarlos p/no volver a tratarlos la siguiente vez). Existe otra posibilidad de q el hilo se salga del `wait()` sin q haya datos disponibles, pero esto se analizará posteriormente.

CONCURRENCIA EN JAVA CON WAIT() Y NOTIFY() - WAIT() Y NOTIFY() COMO COLA DE ESPERA

Wait & Notify
in Java



```
synchronized(lista);  
{  
    lista.add(dato);  
    lista.notify();  
}
```

El hilo q mete datos en la lista tiene q llamar a lista.notify(), p/ esto **es necesario apropiarse del objeto lista con synchronized.**

1 vez hecho esto, **el hilo q estaba bloqueado en wait() despierta, sale del wait()** y seguirá su código leyendo el primer dato de la lista.



WAIT() Y NOTIFY() COMO COLA DE ESPERA

wait() y notify() funcionan como 1 lista de espera. Si varios hilos llaman a wait() s bloquean en 1 lista de espera (1ro.q llamó a wait() es el 1ro.d la lista y el últ.es el últ).

C/llamada a notify() despierta al 1er.hilo en la lista de espera, pero no al resto, q siguen dormidos. Se necesita hacer tantos notify() como hilos hayan hecho wait() p/ir despertándolos a todos de 1 en 1.

Si se hace varios notify() antes de q haya hilos en espera, quedan marcados todos esos notify(), d forma q los sig.hilos q hagan wait() no quedan bloqueados.

wait() y notify() funcionan como un contador. Cada wait() mira el contador y si es cero o menos se bloquea. Cuando s desbloquea decrementa el contador. C/notify() incrementa el contador y si se hace 0 o positivo, despierta al primer hilo de la cola.

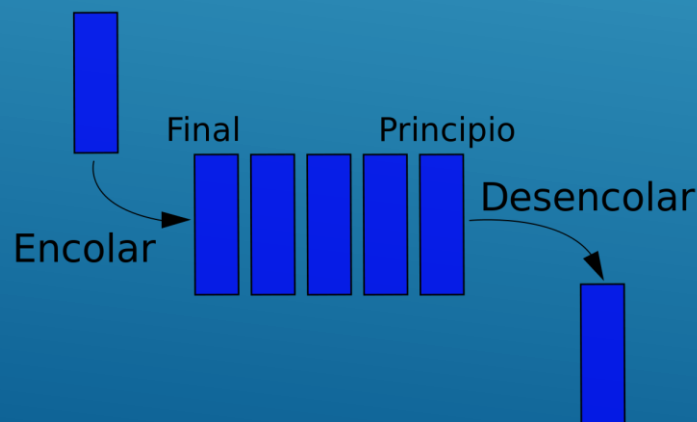


WAIT() Y NOTIFY() COMO COLA DE ESPERA

Comparación: en 1 mesa hay personas, unos ponen caramelos y otros los recogen:

- las personas son hilos,
- los q recogen los caramelos (hacen wait()) se ponen en 1 cola delante de la mesa, recogen 1 caramelo y se van. Si no hay caramelos, esperan q los haya y forman 1 cola;
- otras personas ponen 1 caramelo en la mesa (hacen notify()).

El nro.de caramelos en la mesa es el contador q se mencionó.



MODELO PRODUCTOR/CONSUMIDOR



Es buena costumbre de Orient.a Objetos "ocultar" la sincronización a los hilos, de forma q no se dependa de q el programador se acuerde de implementar su hilo correctamente (llamada a synchronized y llamada a wait() y notify()).

Es conveniente poner la lista de datos dentro de 1 clase y poner 2 métodos synchronized p/añadir y recoger datos, con el wait() y el notify() dentro.

```
public class MiListaSincronizada
{ private LinkedList lista = new LinkedList();
  public synchronized void addDato(Object dato)
  { lista.add(dato);
    lista.notify(); } }
```

```
public synchronized Object
getDato()
{ if (lista.size()==0)
  wait();
  Object dato = lista.get(0);
  lista.remove(0);
  return dato; } }
```

Con esto s
bloquea
hasta q
haya
algún dato
disponible

El hilo que guarda datos sólo debe q hacer: listaSincronizada.addDato(dato);



INTERRUMPIR UN HILO

1 hilo puede salir del wait() sin necesidad d un notify(). Esto es x 1 interrupción, en Java se realiza con el **mét. interrupt()** del hilo. X ej., si el hiloLector está bloqueado en 1wait() esperando 1dato, se puede interrumpir con: **hiloLector.interrupt();**

El hiloLector saldrá del wait() y s encuentra con q no hay datos en la lista. Sabrá q alguien le ha interrumpido y hará lo q tenga que hacer en ese caso.

Ej: 1 hilo lectorSocket pendiente d 1 socket (conexión con otro progr. en otra comput. a través de red):

- 1. Lee datos q llegan del otro progr. y los mete en listaSincronizada.**
- 2. Si existe 1 hilo lectorDatos q lee esos datos de la listaSincronizada y los trata, ¿q ocurre si el socket se cierra?. El progr. cierra la conexión (socket) con el otro programa en red y ya no se comunica con él. Una vez cerrada la conexión, el hilo lectorSocket puede interrumpir al hilo lectorDatos, al ver q ha salido del wait() y q no hay datos disponibles, puede suponer q se ha cerrado la conexión y terminar.**



INTERRUMPIR UN HILO (CÓDIGO EJ.)

```
while (true)
{ if (listaSincronizada.size() == 0)
    wait();
  if (listaSincronizada.size() > 0)           // Se debe comprobar q hay datos
  {      Object dato=listaSincronizada.get(0); // Si hay datos, se tratan
    listaSincronizada.remove(0);
    // tratar el dato. }
  else // Si no hay datos es xq se cerró la conexión, por lo q se sale
    {return; } }
```

y cuando el hilo lectorSocket cierre la conexión debe:

```
socket.close();
```

```
lectorDatos.interrupt();
```





```
public class Main {  
    public static void main(String[] args) {  
        Saludo s = new Saludo(); // Objeto en común, se encarga del wait y notify
```

```
        Personal Empleado1 = new Personal("Pepe", s, false);  
        Personal Empleado2 = new Personal("José", s, false);  
        Personal Empleado3 = new Personal("Pedro", s, false);  
        Personal Jefe1 = new Personal("JEFE", s, true);
```

```
        /*Instancio los hilos y paso como parámetros:
```

```
            * Nombre del Hilo
```

```
            * Objeto en común (Saludo)
```

```
            * Booleano p/verificar si es jefe o empleado */
```

```
        Empleado1.start(); //Lanzo los hilos
```

```
        Empleado2.start();
```

```
        Empleado3.start();
```

```
        Jefe1.start();
```

```
    }
```

```
}
```



EJEMPLO DE WAIT() Y NOTIFY – CLASE PERSONAL

```
import java.util.logging.Level; import java.util.logging.Logger;
public class Personal extends Thread{
    String nombre;
    Saludo saludo;
    boolean esJefe;
    public Personal(String nombre, Saludo salu, boolean esJefe){
        this.nombre = nombre;
        this.saludo = salu;
        this.esJefe = esJefe;    }
    public void run(){
        System.out.println(nombre + " llegó.");
        try {
            Thread.sleep(1000);
            if(esJefe){    //Verifico si el personal que esta es jefe o no
                saludo.saludoJefe(nombre);
            }else{
                saludo.saludoEmpleado(nombre); }
        } catch (InterruptedException ex) {
            Logger.getLogger(Personal.class.getName()).log(Level.SEVERE, null, ex); } } }
```



EJ.DE WAIT() Y NOTIFY – CLASE SALUDO

```
import java.util.logging.Level;
import java.util.logging.Logger;
public class Saludo {
    public Saludo(){ }
```

/* Si no es jefe, el empleado va a quedar esperando a q llegue el jefe. Se hace wait de el hilo q está corriendo y se bloquea, hasta q s le avise q ya puede saludar*/

```
    public synchronized void saludoEmpleado(String nombre){
        try {            wait();
                        System.out.println("\n"+nombre.toUpperCase() + "-: Buenos días jefe.");
        } catch (InterruptedException ex) {
            Logger.getLogger(Saludo.class.getName()).log(Level.SEVERE, null, ex); }    }
```

//Si es jefe, saluda y luego avisa a los empleados p/q saluden
// El notifyAll despierta a todos los hilos que estén bloqueados

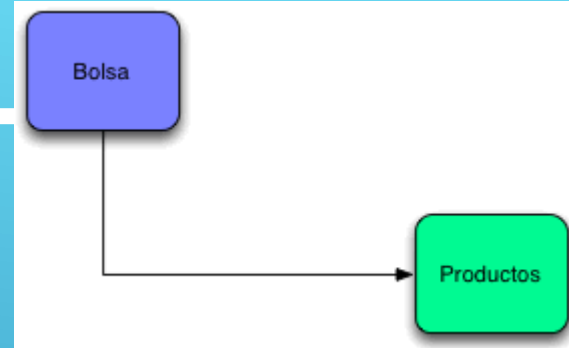

```
    public synchronized void saludoJefe(String nombre){
        System.out.println("\n***** "+nombre + "-: Buenos días empleados. *****");
        notifyAll();    }    }
```

Problema: si el jefe llega antes q 1 empleado éste no despertará de su wait, ya q el notifyAll() ya ha sido llamado x el jefe. Se pide al alumno q plantee 1 solución usando 1 booleano q indique si el jefe ya ha llegado o no, y en función de él saludar al jefe o disculpar el retraso.

EJEMPLO 2 DE WAIT() Y NOTIFY – CLASE BOLSA Y PRODUCTOS

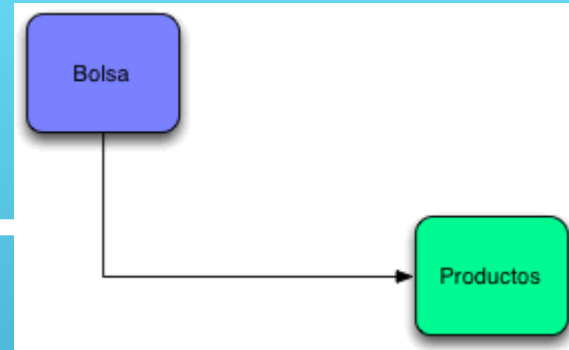
Problema: supongamos que tenemos una bolsa con productos.

```
package com.arquitecturajava;
import java.util.ArrayList;
public class Bolsa {
private ArrayList<Producto> listaProductos = new ArrayList<Producto>();
public void addProducto(Producto producto) {
if (!estaLlena())
    listaProductos.add(producto);
}
public ArrayList<Producto> getListaProductos() {
return listaProductos;
}
public int getSize() {
return listaProductos.size();
}
public boolean estaLlena() {
return listaProductos.size() >= 5;
}
}
```



EJEMPLO 2 DE WAIT() Y NOTIFY – CLASE PRODUCTOS

Problema: supongamos que tenemos una bolsa con productos.



```
package com.arquitecturajava
```



```
public class Producto {
```

```
    private String nombre;
```

```
    public String getNombre() {  
        return nombre;  
    }
```

```
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

EJEMPLO 2 DE WAIT() Y NOTIFY – CLASE HILOENVIO

```
package com.arquitecturajava;
```

```
public class HiloEnvio extends Thread {
```

```
    private Bolsa bolsa;
```

```
public HiloEnvio(Bolsa bolsa) {
```

```
    super();
```

```
    this.bolsa = bolsa; }
```

```
@Override
```

```
public void run() {
```

```
    if (bolsa.estaLlena() != true) {
```

```
        try {
```

```
            synchronized (bolsa) {
```

```
                bolsa.wait(); }
```

```
        } catch (InterruptedException e) { // TODO Auto-generated catch
```

```
        block
```

```
        e.printStackTrace(); }
```

```
        System.out.println("Enviando la bolsa con "+
```

```
        bolsa.getSize()+"elementos"); } }
```

```
public Bolsa getBolsa() { return bolsa; }
```

```
public void setBolsa(Bolsa bolsa) { this.bolsa = bolsa; }
```

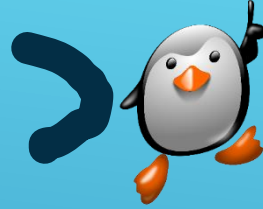
```
}
```



Construimos un Thread que cuando la bolsa este llena la envía. Este Hilo se encarga de sacar por pantalla el mensaje de “Enviando la bolsa con X elementos”. Para ello recibe la Bolsa como parámetro y chequea que esta llena. Usa un bloque de código sincronizado (synchronized). Este bloque de código coordina el trabajo entre nuestros dos Threads y permite que un Thread llene la Bolsa y cuando este llena el otro Thread la envíe.

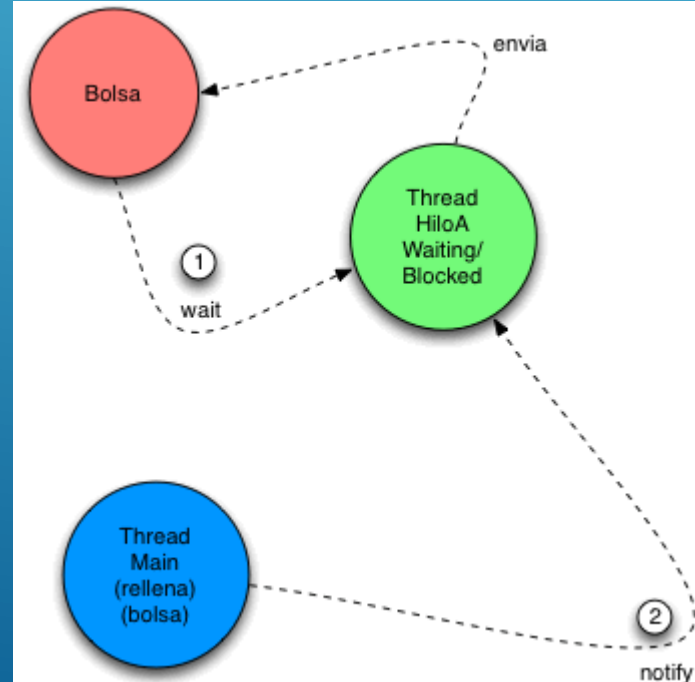
EJEMPLO 2 DE WAIT() Y NOTIFY – CLASE HILOENVIO

```
package com.arquitecturajava;  
public class Principal {  
  public static void main(String[] args) {  
    Bolsa bolsa= new Bolsa();  
    HiloEnvio hilo= new HiloEnvio(bolsa);  
    hilo.start();  
    for(int i=0;i<=10;i++) {  
      Producto p= new Producto();  
      try {  
        synchronized (bolsa) {  
          Thread.sleep(1000);  
          if (bolsa.estaLlena()) {  
            bolsa.notify();  
          }  
        } catch (InterruptedException e) {  
          e.printStackTrace();  
        }  
        bolsa.addProducto(p);  
        System.out.println(bolsa.getSize());  
      }  
    }  
  }  
}
```



Para rellenar la bolsa usamos el Thread del programa principal que cada segundo añadirá un nuevo elemento a la lista.

Ambos Threads deben de estar sincronizados ya que hasta que la bolsa no este llena no la podemos enviar. Para sincronizar el trabajo de los Threads usamos wait notify. HiloEnvio se pondrá a esperar(wait) hasta que la Bolsa este llena antes de enviarla.



BIBLIOGRAFÍA RECOMENDADA



- **García de Jalón J, Rodríguez J, Mingo I, Imaz A, Brazalez A, Larzabal A, Calleja J y García J. 2.000. Aprenda Java como si estuviera en primero.**
- **Páginas web consultadas, accedidas en Septiembre de 2.023:**
 - o http://www.chuidiang.com/java/hilos/wait_y_notify.php
 - o http://www.mundojava.net/excepciones.html?Pg=java_inicial_4_6.html
 - o <http://labojava.blogspot.com.ar/2012/10/ejemplo-de-senalizacion-wait-y-notify.html>
 - o <https://www.arquitecturajava.com/java-wait-notify-y-threads/>