BOLETÍN DE ACTIVIDADES

ACTIVIDAD 1. - Se dispone de un programa en el cual hay un monitor X y 4 tareas T1, T2, T3 y T4. El planificador del sistema emplea un algoritmo de prioridades con prioridades fijas expulsivas, siendo:

Prioridad(T1) > Prioridad (T2) > Prioridad (T3) > Prioridad (T4)

En el instante inicial t0 la tarea T4 está ejecutando un método del monitor X, mientras que las tareas T1, T2 y T3 se encuentran suspendidas. Rellene la siguiente tabla con la planificación del sistema a partir del instante inicial t0, considerando la siguiente secuencia de eventos:

Evento 1) T1 se activa y su siguiente operación consiste en invocar un método del monitor X.

Evento 2) T2 se activa y su siguiente operación es ejecutar código que está fuera del monitor.

Evento 3) El proceso en ejecución invoca una operación del monitor X.

Evento 4) T3 se activa y su siguiente operación es ejecutar código que está fuera del monitor X.

Eventos	**CPU**	**ACTIVOS fuera del monitor X**	**ACTIVOS dentro del monitor X**	**COLA del monitor X**	**Suspendidos**
Inicialmente	T4	---	T4	---	T1,T2,T3
Evento 1)					
Evento 2)					
Evento 3)					
Evento 4)					

ACTIVIDAD 2. - Indique si las siguientes afirmaciones son Verdaderas (V) o Falsas (F). Justifíquelo.

1. Un monitor es un mecanismo de sincronización de alto nivel integrado en algunos lenguajes de programación concurrentes.	
2. El concepto de monitor evita la necesidad de que distintos hilos compartan memoria.	
3. Un monitor es una clase que además resuelve exclusión mutua y sincronización condicional.	
4. Un monitor dispone siempre de una cola de entrada donde esperan aquellos hilos que desean utilizar el monitor cuando lo está utilizando otro hilo.	
5. Dentro de un monitor se pueden producir condiciones de carrera. Cuando esto sucede, el hilo que ejecuta código dentro del monitor debe ejecutar *c.wait()* para abandonarlo.	

ACTIVIDAD 3.- A continuación se muestra la última de las soluciones propuestas en la actividad del problema del *productor-Consumidor* visto en la Unidad 2. Modifíquela apropiadamente, utilizando la sintaxis de Java y el modelo de monitores Java, para que la clase resultante sea un monitor correcto.

```
public class Consumidor extends Thread
{
    private Caja caja;
    private int cnumber;
    public Consumidor(Caja c, int number)
    {
        caja = c;
        cnumber = number;
    }
    public void run()
    {
        int value = 0;
        for (int i = 1; i < 11; i++)
        {
           value = caja.obtener();
           System.out.println("Consumidor #" + cnumber+ " obtiene: " + value);
           try
           {               sleep((int)(Math.random() * 100));           }
           catch (InterruptedException e) { }
        }
    }
}
```

```
public class CondicionDeCarrera
{
    public static void main(String[] args)
    {
        Caja c = new Caja();
        Consumidor c1 = new Consumidor(c, 1);
        Productor p1 = new Productor(c, 1);

        c1.start();
        p1.start();

    }
}
```

```
public class Productor extends Thread
{
    private Caja caja;
    private int prodnumber;
    public Productor(Caja c, int number)
    {
        caja = c;
        prodnumber = number;
    }
    public void run()
    {
        for (int i = 1; i < 11; i++)
        {
            caja.poner(i);
            System.out.println("Productor #" + prodnumber+ " pone: " + i);
            try
            {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
        }
    }
}
```

```
public class Caja
{
   private int contenido =0;
   private boolean llena = false;

   public synchronized int obtener()
   {
        while (!llena) Thread.yield();
        int valor = contenido;
        contenido = 0;
        llena = false;
        return valor;
   }
   public synchronized void poner(int valor)
   {
        while (llena) Thread.yield();
        llena = true;
        contenido = valor;
   }
}
```

ACTIVIDAD 4. Dado el siguiente código, indique si las afirmaciones que se muestran a continuación son Verdaderas (V) o Falsas (F) y justifíquelo.

```
public class BoundedBuffer {
  private int first, last;
  private int numItems,capacity;
  private long items[];
 public BoundedBuffer(int size){
   capacity = size;
   items = new long[size];
   numItems = first = last = 0;
 }

 public synchronized void put(long item){
 if (numItems == capacity)
   try{wait();} catch(Exception e){};
 items[last] = item;
 last = (last + 1) % capacity;
 numItems++;
 notify();
 }
 public synchronized long get() {
 long valor;
 if (numItems == 0)
   try{wait();} catch(Exception e){};
 valor = items[first];
 first = (first + 1) % capacity;
 numItems--;
 notify();
 return valor;
 }
}
```

```
class ProdCons {
  static BoundedBuffer buf=
    new BoundedBuffer(1);

 public static void main(String[] args){
  new Thread(new Runnable() {
      public void run() {
      for (int i=0; i<10; i++) {
          buf.put(i);
  }}},"producer").start();

  new Thread(new Runnable() {
      public void run() {
      for (int i=0; i<10; i++) {
        System.out.println(buf.get());
  }}},"consumer").start();
 }
}
```

1. Al ejecutar este programa se crearán 10 hilos productores y 10 hilos consumidores.	
2. Este código no es correcto pues en Java debe utilizarse siempre `notifyAll()` en lugar de `notify()`.	
3. Este código no funciona, pues al llamar a `wait()` no se dejaría abierto el monitor y nadie más podría acceder a él.	
4. Este programa no ejecuta ninguno de los hilos generados, pues no se invoca el método `run()` de éstos.	
5. Si los hilos invocaran `Thread.currentThread().getName()` en su método `run()`, obtendrían el nombre interno que les asigna Java por defecto.	

ACTIVIDAD 5. Dado el siguiente código, que muestra un monitor para gestionar el acceso a un aparcamiento de coches que tiene una salida y dos entradas (Norte y Sur). Asuma que los hilos (coches) invocan los métodos `enterX`() y `exit`(), antes y después de acceder al aparcamiento respectivamente. Indique si las afirmaciones que se muestran a continuación son Verdaderas (V) o Falsas (F) y justifíquelo.

```
public class gestorAparcamiento {
  private int n = 100;
  private int nCoches = 0;
  private int nEspNorte = 0;
  private int nEspSur = 0;
  private boolean turnS = true;
  private boolean turnN = true;

 public synchronized void entrarSur(){
  nEspSur++;
  while(nCoches == n || !turnS) wait();
  nEspSur--;
  nCoches ++;
  if (nCoches == n) {turnS = false; turnN = true;}
 }

 public synchronized void entrarNorte(){
  nEspNorte++;
  while(nCoches == n || !turnN) wait();
  nEspNorte--;
  nCoches ++;
  if (nCoches == n) {turnS = true; turnN = false;}
 }

 public synchronized void salir(){
  if (nCoches < n ||
     (nCoches == n  && (nEspNorte == 0 || nEspSur == 0 ))){
             turnN = true;
             turnS = true;
  }
  nCoches --;
  notifyAll();
 }
}
```

1. El monitor limita el grado de concurrencia al aparcamiento a 100 coches.	
2. No pueden usar el aparcamiento simultáneamente coches que entren por el norte y coches que entren por el sur.	
3. El primer coche que entre en el aparcamiento habrá utilizado la entrada norte.	
4. El atributo n cuenta el número de coches que están en el aparcamiento.	
5. Cuando el aparcamiento está lleno y hay coches esperando en las dos entradas, se regula el acceso para que cuando hayan plazas libres, se vayan alternando los coches de ambas entradas.	
6. En la cola asociada a `wait`, no pueden haber esperando, simultáneamente, coches que quieren entrar por el norte y coches que quieren entrar por el sur.	

ACTIVIDAD 6. Un taller de una joyería quiere montar collares de perlas solo blancas, solo azules o combinados de perlas blancas y azules. Para ello dispone de 5 encargados y 2 cestos, uno para cada color de perla, con capacidad limitada . Para organizar la producción se decide que un encargado será el proveedor de perlas blancas, otro encargado proveerá las perlas azules y el resto de encargados se destinará al montaje de cada tipo de collar. El monitor `GestorDePerlas`, gestiona el número de perlas almacenadas en los cestos. Hay un hilo asociado a cada encargado. Los encargados de proveer perlas, se encargan de obtener una perla y almacenarla en el cesto correspondiente utilizando los métodos `AñadirBlanca` ó `AñadirAzul`. El resto de encargados solicitan al monitor el número de perlas de cada color que necesitan para montar el collar utilizando el método `SolicitarPedido`. Analice la siguiente propuesta para el monitor **`GestorDePerlas`**.

```
public class GestorDePerlas {

      final static private int NMaxBlancas  = 50;
      final static private int NMaxAzules = 50;

      private int NBlancas  = 0;
      private int NAzules = 0;

      private boolean PedidoEnCurso = false;

      public synchronized void AñadirBlanca() {
            NBlancas = NBlancas ++;
            notifyAll();
            while (NBlancas == NMaxBlancas){
              try {wait();} catch (InterruptedException e){
                            Thread.currentThread().interrupt();}}
      }

      public synchronized void AñadirAzul() {
            NAzules = NAzules ++;
            notifyAll();
            while (NAzules == NMaxAzules){
              try {wait();} catch (InterruptedException e){
                            Thread.currentThread().interrupt();}}
      }

      public synchronized void SolicitarPedido(int SolBlancas,
         int SolAzules) {

            while (PedidoEnCurso){
               try {wait();}
                    catch (InterruptedException e){
                            Thread.currentThread().interrupt();} };

            PedidoEnCurso = true;

            while (SolBlancas > NBlancas || SolAzules > NAzules){
              try {wait();} catch (InterruptedException e){
                            Thread.currentThread().interrupt();}};

            NBlancas  = NBlancas  - SolBlancas;
            NAzules = NAzules - SolAzules;
            PedidoEnCurso = false;
            notifyAll();
      }
}
```

Dado el código anterior, indique si las afirmaciones que se muestran a continuación son Verdaderas (V) o Falsas (F) y justifíquelo.

1.	En esta solución se puede sobrepasar el máximo número de piezas blancas o azules en los cestos, puesto que se incrementan los contadores antes de comprobar si caben.	
2.	El atributo `PedidoEnCurso` es necesario para proporcionar exclusión mútua en el acceso al método `SolicitarPedido`.	
3.	La solución no es correcta porque la invocación al método `notifyAll` en `AñadirBlanca` y `AñadirAzul`, debería ser la última instrucción en ambos métodos.	
4.	El calificativo `synchronized` en los métodos `AñadirBlanca` y `AñadirAzul`, no es necesario ponerlo, ya que sólo hay un hilo que añade piezas blancas y un hilo que añade piezas azules.	
5.	El atributo `PedidoEnCurso` se utiliza para conseguir que cuando un pedido P1 está esperando a que se completen las piezas solicitadas, los nuevos pedidos no se atenderán hasta que se complete P1.	
6.	La solución propuesta para el monitor es correcta, y sincroniza adecuadamente según el enunciado, los proveedores de perlas y la gestión de los pedidos que realizan los montadores.	

ACTIVIDAD 7. Indique si las siguientes afirmaciones son Verdaderas (V) o Falsas (F) y justifíquelo.

1.	El monitor tipo Hoare garantiza que tras una operación *notify()* el hilo reactivado encuentra el estado del monitor exactamente igual que estaba cuando se ejecutó dicho *notify*.	
2.	El monitor tipo Lampson-Redell utiliza una cola especial prioritaria sobre la entrada, donde esperan aquellos que han ejecutado *notify*().	
3.	Un monitor que siga el modelo de Hoare suspende al hilo que invoca a *notify*(), quedándose dicho hilo suspendido en una cola especial.	
4.	Un monitor que siga el modelo de Lampson/Redell jamás suspende a un hilo en la invocación a *notify*().	
5.	El lenguaje Java proporciona por defecto monitores de tipo "Lampson/Redell".	

ACTIVIDAD 8.- En una calzada por la que circulan **coches** se tiene un paso de **peatones** cuya condición de corrección es que coches y peatones no pueden cruzarlo simultáneamente. El paso de peatones está gobernado por el monitor que se presenta seguidamente. Los métodos ***enterX()*** son invocados por los hilos de tipo X (C=Coche o P=Peatón) al llegar al paso de peatones. Los métodos ***leaveX()*** son invocados por los hilos de tipo X al abandonar el paso de peatones.

```
monitor Crosswalk {
   condition OKcars, OKpedestrians;
   int c, c_waiting, p, p_waiting;

   public Crosswalk() {
     c = c_waiting = p = p_waiting = 0;
   }

   entry void enterC() {
     c_waiting++;
     while ( COND-1 ) OKcars.wait();
     c_waiting--;
     c++;
     OKcars.notify();
   }

   entry void leaveC() {
       c--;
       OKcars.notify();
       OKpedestrians.notify();
   }

   entry void enterP() {
       p_waiting++;
       while ( COND-2 ) OKpedestrians.wait();
       p_waiting--;
       p++;
       OKpedestrians.notify();
   }

   entry void leaveP() {
         p--;
         OKcars.notify();
         OKpedestrians.notify();
   }
}
```

Suponiendo un monitor M de esa clase Crosswalk, implantado siguiendo la variante de **Lampson/Redell** (que implica que cuando un hilo suspendido en una condición es reactivado entonces pasa a la cola de entrada), y el siguiente valor de las expresiones COND-1 y COND-2:

COND-1 ≡ (p >0) y COND-2 ≡ (c>0) || (c_waiting >0)

a) Describa la evolución del estado de cada uno de los atributos del monitor si se produce la siguiente secuencia ordenada de invocaciones a métodos del monitor.

	Método invocado	c_waiting	c	Cola OKcars	p_waiting	p	Cola OKpedestrians
0)	(inicial)	0	0	vacía	0	0	vacía
1)	P1:M.enterP();						
2)	P2:M.enterP();						
3)	C1:M.enterC();						
4)	P3:M.enterP();						
5)	C2:M.enterC();						
6)	P1:M.leaveP();						
7)	C3:M.enterC();						
8)	P2:M.leaveP();						
9)	Fin de la traza						

b) Observe que este monitor otorga prioridad a un tipo de hilos. ¿A cuál de las dos? ¿Por qué?

c) Se requiere otorgar ahora prioridad al otro tipo de hilo. Rellene esta tabla explicativa de la sincronización condicional que se necesita. Indique por qué motivos deberían esperar los hilos en cada método, qué estados se modifican si los métodos se ejecutan por completo, y a quiénes se tendría que notificar y por qué motivo.

Crosswalk	**Espera cuando..**	**Modifica estado…**	**Notifica a.. (indicar por qué)**
enterC()			
leaveC()			
enterP()			
leaveP()			

d) Indique qué modificaciones debería realizar sobre el código del monitor para aplicar lo indicado en la tabla anterior.

e) Compruebe a continuación que la solución propuesta es adecuada, completando la siguiente traza. En el instante 10, ¿cuántos peatones están cruzando la calzada? ¿Y cuántos coches están esperando?

	Método invocado	**c_waiting**	**c**	**Cola OKcars**	**p_waiting**	**p**	**Cola OKpedestrians**
0)	(inicial)	0	0	vacía	0	0	vacía
1)	P1:M.enterP();						
2)	P2:M.enterP();						
3)	C1:M.enterC();						
4)	P3:M.enterP();						
5)	C2:M.enterC();						
6)	P1:M.leaveP();						
7)	C3:M.enterC();						
8)	P4:M.enterC();						
9)	P2:M.leaveP();						
10)							

f) Indique cómo podría continuar esta traza para que, de forma correcta, pasaran por la calzada todos los peatones y coches indicados en el ejemplo anterior.

ACTIVIDAD 9.- Se desea implantar mediante monitores un enlace de comunicación de capacidad nula. Esto implica que si llega antes el receptor, tendrá que esperar a que el emisor envíe el mensaje. A su vez, si llegara antes el emisor, esperaría hasta que el receptor recogiera el mensaje, pues el S.O. no usa ningún buffer para mantener temporalmente los mensajes pendientes de entrega. Se ha implantado dicho enlace mediante el monitor que se presenta seguidamente (Obsérvese que los enlaces de capacidad nula solo interconectan a un par de procesos. No tiene sentido asumir más de un emisor ni más de un receptor):

```
monitor SynchronousLink {
  condition OKsender, OKreceiver;
  int senders_waiting, receivers_waiting;
  Message msg;

  public SynchronousLink() {
     senders_waiting = receivers_waiting = 0;
     msg = null;
  }

  entry void send(Message m) {
     if (receivers_waiting > 0) {
        msg = m;
        OKreceiver.notify();
     } else {
        senders_waiting++;
        OKsender.wait();
        senders_waiting--;
        msg = m;
     }
  }

  entry Message receive() {
     if (senders_waiting > 0) {
        OKsender.notify();
     } else {
        receivers_waiting++;
        OKreceiver.wait();
        receivers_waiting--;
     }
     return msg;
  }
}
```

a) Razone si este monitor ofrecerá el comportamiento solicitado en caso de asumir el modelo de Hoare.
b) Razone si este monitor ofrecerá el comportamiento solicitado en caso de asumir el modelo de Lampson y Redell.
c) Implante este monitor utilizando Java para que tenga el comportamiento descrito en el enunciado. **Justifique** si su solución tiene realmente capacidad nula o no.

ACTIVIDAD 10.- En el problema de las hormigas se dispone de varios hilos "hormiga" que quieren moverse de una celda a otra en un mismo territorio, con la restricción que en una misma celda no puede haber más de una hormiga. Sabiendo que el territorio actúa como un monitor, complete el código que se proporciona a continuación, asumiendo las siguientes dos opciones:

Opción 1) Hay una variable condición para cada celda del territorio.

```
Monitor Territory{
      boolean [N][N] occupied;



      entry void moves(int x, y, x', y'){



            occupied[x',y']=true;
            occupied[x,y]=false;



      }
}
```

Opción 2) Hay una única variable condición para todo el territorio.

```
Monitor Territory{
      boolean [N][N] occupied;


      entry void moves(int x, y, x', y'){



            occupied[x',y']=true;
            occupied[x,y]=false;


      }
}
```

Cuestiones:

a) ¿Cuál de estas dos soluciones es más eficiente? ¿Por qué?

b) ¿Cuál requiere "reactivación en cascada" (también llamada "notificación en cascada")?

c) Indique con qué tipo de monitor funcionará correctamente cada una de estas opciones.

ACTIVIDAD 11.- En un bosque conviven 10 lobos y 10 corderos que comparten un río al que acceden para beber. El acceso al río se gestiona por el siguiente código:

```
monitor Rio {
  condition entrar,salir;
  int dentro=0;

  entry void lobo_entrar()
  {
    // lobo tiene hambre y sed
    if ( dentro == 1) comer_cordero();
  }

  entry public void lobo_salir()
  {   // lobo deja de beber  }

  public comer_cordero(){
    //ñam
  }
```

```
  entry void cordero_entrar()
  {  // cordero tiene sed
     entrar.notify();
     if ( dentro == 0) entrar.wait();
     dentro++;
     salir.notify();
   }

  entry void cordero_salir()
  {   salir.notify();
     if ( dentro == 2) salir.wait();
     dentro--;
     // cordero deja de beber
   }
} // fin monitor
```

Asumiendo que tanto los lobos como los corderos respetan siempre el protocolo de invocar el método *lobo/cordero_entrar*(), antes de acceder al río para beber y *lobo/cordero_salir*() cuando abandonan el río, conteste a las siguientes preguntas.

a) Si se utiliza la variante de monitores de Hoare, ¿los lobos y los corderos acceden al río en exclusión mutua? Es decir, ¿si hay lobos bebiendo no puede haber corderos bebiendo y viceversa? ¿Se podrán comer los lobos a los corderos? Justifique sus respuestas.

b) Si la variante de monitor utilizada es la de Lampson Redell, ¿se podrán comer los lobos a los corderos? Justifique su respuesta con una traza.

ACTIVIDAD 12.- En una empresa se utiliza un baño mixto en el que pueden entrar tanto hombres como mujeres, pero con la condición de que simultáneamente sólo pueda haber personas de un único sexo. Además, el baño tiene una capacidad limitada de 3 personas.

```
monitor Baño {
    condition lleno, sexo_opuesto;
    int ocupantes=0, capacidad=3;
    boolean mujeres=FALSE;
    entry entra_adulto(boolean esMujer){
       if (ocupantes+1 > capacidad) lleno.wait();
       if (ocupantes>0 && mujeres!= esMujer){
          lleno.notify();
          sexo_opuesto.wait();
          sexo_opuesto.notify();
     }
    ocupantes++;
    mujeres = esMujer;
   }

   entry sale_adulto(){
         ocupantes--;
         if (ocupantes+1 == capacidad)
               lleno.notify();
         else  if (ocupantes==0)
                  sexo_opuesto.notify();
    }
}
```

Asumiendo que tanto hombres como mujeres invocan a los métodos del monitor según el protocolo *entra_adulto()* ... *sale_adulto()*, y sabiendo que el monitor debe controlar que no se exceda de la capacidad del baño y que solamente puedan usarlo personas del mismo sexo al mismo tiempo, responda a las siguientes preguntas:

a) Con la variante de Hoare, ¿podrían entrar adultos de distinto sexo en el baño? ¿se podría sobrepasar la capacidad del baño? Muestre alguna(s) traza(s) que justifique sus respuestas.

b) Con la variante de Hoare, ¿habría diferencia en la ejecución del monitor si en vez de las sentencias "if" del método entra_adulto tuviéramos sentencias "while"?

c) Realice el mismo análisis para la variante de Lampson-Redell. Es decir, con Lampson-Redell ¿pueden entrar adultos de distinto sexo en el baño? ¿se puede sobrepasar la capacidad del baño? Muestre alguna traza que justifique sus respuestas.

d) ¿Se hace uso en este monitor de la "reactivación en cascada"? Si es así, ¿cómo se habría podido implementar en Java?

ACTIVIDAD 13.- Dado el siguiente programa:

```
public class TestIt {
  public class Test {
    public synchronized void test (Test t) {
      t.SayHola ();
    }
    public synchronized void SayHola () {
      System.out.println ("Hola");
    }
  }

  public TestIt () {
    Test t1 = new Test();
    Test t2 = new Test();
    new Thread (new Runnable () {public void run() {
      t1.test(t2);
    }}).start();
    new Thread (new Runnable () {public void run() {
      t2.test(t1);
    }}).start();
  }
  public static void main (String args[]) {
    new TestIt();
  }
}
```

Responda si las siguientes afirmaciones son verdaderas o falsas, y justifique su respuesta.

1. Si por la pantalla vemos una vez "Hola", seguro que veremos "Hola" dos veces.	
2. Se trata de un programa correcto, libre de condiciones de carrera y libre de interbloqueos, pues los métodos son "synchronized".	
3. En toda ejecución, al menos veremos por la pantalla "Hola", una vez.	
4. Puede presentar condiciones de carrera en el acceso a las variables t1 y t2.	