NOTA: Aquest butlletí d'activitats utilitza en la major part dels exercicis plantejats la notació/sintaxi general per a monitors suggerida en la fulla 12 del material de teoria. Com a única excepció, aquelles activitats que assumisquen el model utilitzat en Java, empraran la sintaxi pròpia de Java. La variant a utilitzar (la descripció de la qual s'ha presentat en les fulles 22 a 26 de teoria) s'indicarà explícitament en l'enunciat.

ACTIVITAT 1

OBJECTIUS: Comprendre el model de programació concurrent proporcionat pels monitors.

ENUNCIAT: Es disposa d'un programa en el qual hi ha un monitor X i 4 activitats T1, T2, T3 i T4. El planificador del sistema empra un algorisme de prioritats amb prioritats fixes expulsives, sent:

Prioritat(T1) > Prioritat (T2) > Prioritat (T3) > Prioritat (T4)

En l'instant inicial t0 l'activitat T4 està executant un mètode del monitor X, mentre que les activitats T1, T2 i T3 es troben suspeses.

Emplene la següent taula amb la planificació del sistema a partir de l'instant inicial t0, considerant la següent seqüència d'esdeveniments:

1. T1 s'activa i la seua següent operació consisteix a invocar un mètode del monitor X.
2. T2 s'activa i la seua següent operació és executar codi que està fora del monitor.
3. El procés en execució invoca una operació del monitor X.
4. T3 s'activa i la seua següent operació és executar codi que està fora del monitor X.

Esdeveniments	**CPU**	**ACTIUS fora del monitor X**	**ACTIUS dins del monitor X**	**CUA del monitor X**	**Suspesos**
Inicialment	T4	---	T4	---	T1,T2,T3
Esdeveniment 1)					
Esdeveniment 2)					
Esdeveniment 3)					
Esdeveniment 4)					

ACTIVITAT 2

OBJECTIUS: Explicar el model de programació concurrent proporcionat pels monitors. Solucionar el problema de la sincronització condicional mitjançant la utilització de monitors.

ENUNCIAT: En una calçada per la qual circulen **cotxes** es té un pas **de vianants** la condició de correcció dels quals és que cotxes i vianants no poden creuar-lo simultàniament. El pas de vianants està governat pel monitor que es presenta seguidament. Els mètodes ***enterX()*** són invocats pels fils de tipus X (C=Cotxe o P=Vianant) en arribar al pas de vianants. Els mètodes ***leaveX()*** són invocats pels fils de tipus X en abandonar el pas de vianants.

```
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();
    }
}
```

Si es considera un monitor M d'aquesta classe Crosswalk, implantat seguint la variant de Lampson/Redell i el següent valor de les expressions COND-1 i COND-2:

$$\text{COND-1} \equiv (p > 0) \quad \text{i} \quad \text{COND-2} \equiv (c > 0) \,||\, (c\_waiting > 0)$$

a) Descriga l'evolució de l'estat de cadascun dels atributs del monitor si es produeix la següent seqüència ordenada d'invocacions a mètodes del monitor. (NOTA: Assumisca que la fila i representa l'estat dels atributs del monitor en l'instant en què s'inicia l'execució de l'operació de la fila i+1. La notació C1:M.enterC() significa que el fil C1 invoca l'operació enterC() del monitor M).

	Mètode invocat	**c_waiting**	**c**	**Cua OKcars**	**p_waiting**	**p**	**Cua OKpedestrians**
0)	(inicial)	0	0	buida	0	0	buida
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)	Fi de la traça.						

b) Observe que aquest monitor atorga prioritat a un tipus de fils. A quin dels dos? Per què? (És a dir, raone quin principi ha de seguir-se per a concedir major prioritat a un determinat tipus de fil).

c) Indique quines modificacions hauria de realitzar sobre el codi del monitor perquè tinguera prioritat l'altre tipus de fil.

ACTIVITAT 3

OBJECTIUS: Avaluar les variants de monitor existents.

ENUNCIAT: Reescriga el codi del monitor de l'Activitat 2 perquè implante el model de Brinch Hansen. Recorde que en el model de Brinch Hansen només pot haver-hi una crida al notify() d'una determinada condició com a última sentència d'aquells mètodes on haja d'utilitzar-se notify(). És a dir, no poden executar-se dos notify() seguits com en l'exemple de l'Activitat 2.

ACTIVITAT 4

OBJECTIUS: Avaluar les variants de monitor existents. Construir monitors en llenguatges de programació concurrent, evitant els seus problemes potencials.

ENUNCIAT: Es desitja implantar mitjançant monitors un enllaç de comunicació de capacitat nul·la. Açò implica que si arriba abans el receptor, haurà d'esperar que l'emissor envie el missatge. Al seu torn, si arribara abans l'emissor, esperaria fins que el receptor arreplegara el missatge, ja que el S.O. no usa cap buffer per a mantenir temporalment els missatges pendents de lliurament. S'ha implantat aquest enllaç mitjançant el monitor que es presenta seguidament (Observe's que els enllaços de capacitat nul·la solament interconnecten a un parell de processos. No té sentit assumir més d'un emissor ni més d'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) Raone si aquest monitor oferirà el comportament sol·licitat en cas d'assumir el model de Brinch Hansen.

b) Raone si aquest monitor oferirà el comportament sol·licitat en cas d'assumir el model de Hoare .

c) Raone si aquest monitor oferirà el comportament sol·licitat en cas d'assumir el model de Rampson i Redell.

d) Implante aquest monitor utilitzant Java perquè tinga el comportament descrit en l'enunciat. **Justifique** si la seua solució té realment capacitat nul·la o no.

ACTIVITAT 5

OBJECTIUS: Construir monitors en llenguatges de programació concurrent, evitant els seus problemes potencials.

ENUNCIAT: Revise les propostes de solució que es van proporcionar en l'enunciat de l'Activitat 4 de la Unitat 2.

A) A continuació es mostra la solució 4 d'aquesta activitat. Modifique-la apropiadament, utilitzant la sintaxi de Java i el model de monitors Java, perquè la classe resultant siga un monitor correcte.

```
public class Productor extends Thread
{
    private Caixa caixa;
    private int prodnombre;
    public Productor(Caixa c, int nombre)
    {
         caixa = c;
         prodnombre = nombre;
    }
    public void run()
    {
         for (int i = 1; i < 11; i++)
         {
             caixa.ficar(i);
             System.out.println("Productor #" + prodnombre + " fica: " + i);
             try
             {
                 sleep((int)(Math.random() * 100));
             } catch (InterruptedException e) { }
         }
    }
}
```

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

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

     }
}
```

```
public class Caixa
{
   private int contingut =0;
   private boolean plena = false;

   public synchronized int obtenir()
   {
         while (!plena) Thread.yield();
         int valor = contingut;
         contingut = 0;
         plena = false;
         return valor;
   }
   public synchronized void ficar(int valor)
   {
         while (plena) Thread.yield();
         plena = true;
         contingut = valor;
   }
}
```

```
public class Consumidor extends Thread
{
     private Caixa caixa;
     private int cnombre;
     public Consumidor(Caixa c, int nombre)
     {
         caixa = c;
         cnombre = nombre;
     }
     public void run()
     {
         int valor = 0;
         for (int i = 1; i < 11; i++)
         {
            valor = caixa.obtenir();
            System.out.println("Consumidor #" + cnombre + " obté: " + valor);
            try
            {               sleep((int)(Math.random() * 100));            }
            catch (InterruptedException e) { }
         }
     }
}
```

B) Esculla una altra de les propostes de solució i modifique-la apropiadament, utilitzant Java i el model de monitors de Java, perquè la classe resultant siga un monitor correcte.

ACTIVITAT 6

OBJECTIUS: Construir monitors en llenguatges de programació concurrent, evitant els seus problemes potencials.

ENUNCIAT: Estudie el codi que es mostra en les figures següents (està disponible en PoliformaT com BoundedBuffer.zip, en la carpeta associada al "Material Aula" de la UD03). Aquest codi pretén resoldre el problema del buffer de capacitat limitada utilitzant dues "condicions" en Java. Aquesta activitat pretén justificar per què no és possible realitzar-ho de la manera ací plantejada (es veuran els problemes potencials).

Descarregue el fitxer ".zip" que s'ha esmentat i comprove el seu funcionament. Realitze algunes modificacions sobre les classes Main (generant més productors i consumidors, per exemple) i BoundedBuffer (qualificant els seus mètodes put() i get() amb "synchronized", per exemple). Observe els resultats d'aquestes modificacions executant el programa resultant. Conteste posteriorment a les qüestions que es presenten al final.

```
public class BoundedBuffer {
    private long numItems;
    private int first,last,capacity;
    private long items[];
    private Object notFull;
    private Object notEmpty;

    public BoundedBuffer(int size){
       capacity = size;
       items = new long[size];
       numItems = first = last = 0;
       notFull = new Object();
       notEmpty = new Object();
    }
    public void put(long item) {
        if (numItems == capacity) try {
                synchronized(notFull) {
                    notFull.wait();
                }
             }catch(Exception e){};
       items[last] = item;
       last = (last + 1) % capacity;
       numItems++;
       synchronized(notEmpty){
           notEmpty.notify();
       }
    }
```

```
    public long get() {
       long valor;
       if (numItems == 0) try {
               synchronized(notEmpty){
                   notEmpty.wait();
               }
           }catch(Exception e){};

       valor = items[first];
       first = (first + 1) % capacity;
       numItems--;
       synchronized(notFull){
           notFull.notify();
       }
       return valor;
    }
}
```

```
public class Main {
    static public void main(String[] args) {
        BoundedBuffer buf = new BoundedBuffer(4);
        Consumer c = new Consumer(buf,1);
        Producer p = new Producer(buf,1);

        p.start();
        c.start();
    }
}
```

```java
public class Consumer extends Thread {
    private BoundedBuffer buf;
    private int id;

    public Consumer( BoundedBuffer buffer, int ident ) {
        buf = buffer;
        id = ident;
    }

    public void run() {
        int i;
        long el;

        for (i=0; i<10; i++) {
            System.out.println("Consumer "+id+
                               ": obtaining element " +
                               "...");
            el = buf.get();
            System.out.println("Consumer "+id+
                               ": Element "+el+".");
        }
        System.out.println("End of consumer "+id+".");
    }
}
```

```java
public class Producer extends Thread {
    private BoundedBuffer buf;
    private int id;

    public Producer( BoundedBuffer buffer, int ident ) {
        buf = buffer;
        id = ident;
    }

    public void run() {
        long i;

        for (i=0; i<10; i++) {
            System.out.println("Producer "+id+
                               ": inserting element "+i+
                               "...");
            buf.put(i);
            System.out.println("Producer "+id+": OK!");
        }
        System.out.println("End of producer "+id+".");
    }
}
```

Conteste aquestes qüestions sobre la classe BoundedBuffer:

a) Implanta correctament la sincronització condicional? Per què? Necessitaríem canviar els “if” d'aquestes sentències per “while”?

b) Necessitem etiquetar els mètodes put() i get() de la classe BoundedBuffer amb l'etiqueta “synchronized”? Què implicaria aquest canvi?

c) Justifique si pot considerar-se que la classe BoundedBuffer, tal qual s'ha definit inicialment en l'activitat, és un monitor correcte. De no ser així indique per què i proporcione una possible traça en la qual es genere una “condició de carrera”.