EXAMEN PRÁCTICA 2 - SCD - (14/12/14) - CARLOS UREÑA
SINCRONIZACIÓN CON MONITORES EN JAVA (EXAMEN TIPO B)

El tiempo de realización del examen es de 60 minutos. La primera parte se responderá contestando en el propio folio del examen. La segunda parte se realiza partiendo de la solución proporcionada por el alumno a la práctica. Se debe subir el código generado a Tutor antes del plazo que el profesor indique.

1. Corrección de errores (6 puntos)

Accede a Tutor y descarga el archivo que el profesor te ha proporcionado (A1-ebs.java). El código que contiene trata de dar solución al problema del Barbero Durmiente usando un monitor abstracto tipo *hoare*. Sin embargo, hay diversos errores en el código proporcionado que deberás solucionar.

1a.1. Intenta compilar el código proporcionado. Comprueba que hay diversos errores básicos de sintaxis que impiden que el código genere un ejecutable. Encuentra dichos errores y arréglalos hasta que el código compile correctamente. Indica en el folio del examen el número de línea y cómo debería quedar ésta una vez solventado el problema.

1a.2. Ejecuta el programa y observa la salida que genera. El algoritmo presenta un error en su diseño. Explica brevemente en el folio del examen por qué la salida proporcionada no es correcta.

1a.3. Observa el código del programa ejecutado. Encuentra y corrige el error en el diseño del algoritmo que impide que el programa muestre una salida coherente. Señala en el folio del examen en qué línea está el problema y cómo debería quedar ésta para solucionar el error.

1a.4. Realiza los cambios necesarios en el algoritmo para que al llegar un cliente nuevo a la barbería, si el barbero está ocupado pelando a otro cliente y la sala de espera no está vacía, entonces el cliente salga a esperar a la calle en lugar de sentarse en la sala de espera. Para representar la espera en la calle, basta indicar a la hebra que duerma un tiempo aleatorio y luego vuelva a entrar.

2. Problema de de los Fumadores usando un Monitor Abstracto (4 puntos)

Partiendo del algoritmo desarrollado durante la Práctica 2 para resolver el problema de los Fumadores usando una implementación de monitor abstracto tipo *hoare*, se pide modificar el algoritmo para que el estanco tenga dos estanqueros y 6 fumadores. Dos fumadores necesitarán cerillas, dos necesitarán papel y los otros dos necesitarán tabaco. Los estanqueros generarán en bucle uno de estos tres ingredientes en bucle como hasta ahora sirviendo a los 6 fumadores.

Por otra parte, deberemos ajustar las hebras estanqueras para que cualquiera de los dos estanqueros pueda generar un nuevo ingrediente especial (Identificado por ejemplo, con el número 4) que representará la "Hora de cerrar el estanco". Este ingrediente se podrá generar al azar en cualquier momento de igual manera que se generan los demás. Cuando este ingrediente se genere, se deberán finalizar a todas las hebras fumador y a continuación los dos estanqueros correctamente, mostrando un mensaje en pantalla informando de cuando finaliza cada una.

```
import monitor.* ;
import java.util.Random ;

class auxb
{
   static Random genAlea = new Random() ;
   static void dormir_max( int milisecsMax )
   {  try
      { Thread.sleep( genAlea.nextInt( 200+milisecsMax ) ) ;
      }
      catch( InterruptedException e )
      { System.err.println("sleep interumpido en 'aux.dormir_max()'");
      }
   }
   static public void mensaje( String str )
   {
      System.out.println( Thread.currentThread().getName()+": "+str);
   }
}
class Barberia extends AbstractMonitor
{
   Condition clientesEnEspera,
             barberoDormido ,
             clienteEnCorte ;

   public Barberia()
   {
      clientesEnEspera = makeCondition() ;
      clienteEnCorte  = makeCondition() ;
      barberoDormido  = makeCondition() ;
   }
   public void cortarPelo()
   {
      enter() ;

      auxb.mensaje("llega a la barberia.");
      if ( barberoDormido.isEmpty() )
         clientesEnEspera.await() ;
      else
         barberoDormido.signal() ;
      auxb.mensaje("comienza a cortarse el pelo.");
      clienteEnCorte.await() ;


      auxb.mensaje("ha terminado de cortarse el pelo.");
      leave() ;

   }
   public void siguienteCliente()
   {
      enter() ;
      auxb.mensaje("siguiente cliente.");
      clientesEnEspera.signal() ;
      auxb.mensaje("comienza a cortar el pelo");
      leave() ;
   }
   public void finCliente()
   {
      enter() ;
      auxb.mensaje("finaliza corte de pelo, indicará al cliente que salga");
      clienteEnCorte.await() ;
      leave() ;
   }
}
class Cliente implements Runnable
{
   private Barberia    barberia ;
   public  Thread      thr ;
```

```
   public Cliente( Barberia p_barberia, int p_contador )
   {
      barberia = p_barberia ;
      thr      = new Thread(this,"cliente "+p_contador);
   }
   public void run()
   {
      while( true )
      {
         barberia.cortarPelo() ;

             auxb.mensaje("El cliente sale de la barbería");
             auxb.dormir_max( 1000 ) ;

      }
   }
}
class Barbero implements Runnable
{
   int       barberia ;
   public Thread  thr ;

   public Barbero( Barberia p_barberia )
   {
      barberia = p_barberia ;
      thr      = new Thread(this,"barbero");
   }
   public void run()
   {
      while( true )
      {
         barberia.siguienteCliente() ;
         auxb.dormir_max( 1000 );
         barberia.finCliente() ;
      }
   }
}
class A1ebs
{
   public static void main( String args[] ) throws InterruptedException
   {
      System.out.println("comienza simulación de barbería.");

      Barberia  barberia = new Barberia() ;
      Barbero   barbero  = new Barbero();
      Cliente[] cliente  = new Cliente[10] ;

      for( int i = 0 ; i < 10 ; i++ )
        cliente[i] = new Cliente(barberia,i) ;

      barbero.thr.start() ;
      for( int i = 0 ; i < 10 ; i++ )
        cliente[i].thr.start() ;

      barbero.thr.join() ;
      for( int i = 0 ; i < 10 ; i++ )
         cliente[i].thr.join() ;

      System.out.println("simulación de barbería finalizada normalmente.");
   }
}
```

SOLUCIÓN (ERRORES COMPILACIÓN)

1. Línea 89. Barbería no es un objeto de tipo int, barbería debe ser un objeto de tipo Barberia para poder inicializar la clase Barbero.

   Barberia barberia ;

2. Línea 114. Falta el argumento Barberia para poder inicializar el objeto barbero.

   Barbero barbero = new Barbero(barberia);

SOLUCIÓN (ERROR DISEÑO)

El error en la salida está en que cuando el barbero termina de cortar el pelo al cliente es nunca sale de la barbería. Debe haber algún error a la hora de despertar a la hebra Cliente para indicarle que ha terminado su corte de pelo.

Línea 61. Tras acabar de cortar el pelo, en lugar de despertar al cliente para indicarle que salga se le está volviendo a indicar que duerma (await). Se debería hacer un signal a la hebra para despertarla.

clienteEnCorte.signal() ;