**UNIVERSIDAD DE LOS ANDES**
**FACULTAD DE INGENIERÍA**
**DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN**
**INFRAESTRUCTURA COMPUTACIONAL - ISIS 2203**
**Primer parcial 2014-2 - SOLUCIÓN**

## A. (70%) THREADS Y SINCRONIZACIÓN EN JAVA

Se desea implementar un buffer con comunicación sincrónica. Similar al caso 1, tendremos varios productores que dejan sus mensajes en un buffer de tamaño limitado; en este caso el tamaño del buffer es igual al número de productores, y solo hay 1 consumidor. En el buffer, los mensajes serán números enteros almacenados en un *wrapper* de tipo `Integer`.

**Funcionamiento**:

Productor:

- Genera un mensaje, y lo deposita en el buffer; esto siempre es posible dado que se trata de comunicación sincrónica, y el buffer es igual al número de productores.
- El número de mensajes se pasa como parámetro en el constructor del productor; cuando acaba de enviarlos, el productor invoca el método `void retirarCliente()` de `Buffer` para indicar que se retira.

Consumidor:

- Continuamente intenta retirar mensajes del buffer; si está vacío, continúa intentando en espera activa.
- Para saber si quedan clientes usa el método `boolean hayClientes()` de `Buffer`.

Buffer:

- El número de productores se pasa como parámetro en el constructor del buffer.
- Los productores acceden al buffer por medio del método `void enviar(int)`, cuyo parámetro es el mensaje que se desea enviar.
- Una vez almacenado el mensaje, el productor se duerme a la espera de que el mensaje sea retirado por el consumidor (ya que se trata de comunicación sincrónica); solo retorna cuando el mensaje sea retirado.
- Cuando el productor acaba de enviar sus mensajes, invoca el método `void retirarCliente()` de `Buffer` para indicar que se retira.
- En cuanto al consumidor, accede al buffer por medio del método `Integer recibir()`.
- Si buffer está vacío, `recibir` retorna `null`; si no, el primer mensaje en el buffer.
- Cuando extrae un mensaje, `recibir` solo despierta al productor que lo depositó.
- El método `boolean hayClientes()` indica si todos los productores ya acabaron.

Note que los productores deben esperar sobre los `Integer` almacenados en el `ArrayList`.

Este esquema se implementará usando 4 clases: Principal, Productor, Consumidor y Buffer. Completar el esqueleto que se encuentra a continuación, para esto:

- Complete el `main`, los encabezados, atributos y constructores de todas las clases.
- Escriba los métodos del productor y el consumidor.
- Escriba `enviar`, `recibir`, `retirarCliente` y `hayClientes` en la clase buffer.

**// Principal: Complete encabezado y main**

```
public class Principal {

    private static final int N_PRODUCTORES = un-cierto-valor;
    private static Buffer buffer;
    public static void main(String[] args) {

      buffer = new Buffer( N_PRODUCTORES );

      for ( int i = 0; i < N_PRODUCTORES; i++ )
        new Productor( buffer, numero-mensajes ).start( );

      new Consumidor( buffer ).start( );

    }

}
```

**// Productor: Complete encabezado, atributos y métodos**

```
public class Productor extends Thread {

  private int nMensajes;
  private Buffer buffer;

  public Productor( Buffer b, int n ){

    nMensajes = n;
    buffer = b;

  }

  public void run( ){

    for ( int i = 0; i < nMensajes; i++ )
      try {
        buffer.enviar( i );
      } catch ( InterruptedException e ){ };

    buffer.retirarCliente( );

  }

}
```

**// Consumidor : Complete encabezado, atributos y métodos**

```
public class Consumidor extends Thread {

  private Buffer buffer;

  public Consumidor( Buffer b ){

    buffer = b;
  }

  public void run( ){

    Integer m = null;

    while ( b.hayClientes( ) ) {
      while ( buffer.hayClientes( )
              && ( m = buffer.recibir( ) ) == null );
      procesar-m;
    }
  }

}
```

**// Buffer: Complete encabezado y métodos**

```
public class Buffer {

     private int nProductores;
     private int tamañoBuffer;
     private ArrayList mensajes;

  public Buffer( int nClientes ){

    tamañoBuffer = nClientes;
    nProductores = nClientes;
    mensajes = new ArrayList( );

  }

  public void enviar( int n ) throws InterruptedException {

    Integer m = new Integer( n );

    synchronized ( m ){
      synchronized ( this ){
        mensajes.add( m );
      }
      m.wait();
    }

  }


  public synchronized Integer recibir(){
```

```
    Integer m;

    if ( mensajes.size( ) > 0 ){
      m = (Integer)mensajes.remove( 0 );
      synchronized ( m ) { m.notify( ); }
      return m;
    }
    else return null;

  }

  public boolean hayClientes( ) {

    return nClientes != 0;

  }

  public void retirarCliente( ) {

    nClientes--;

  }

}
```