Trabajo práctico Nº 6

Mecanismos de sincronización:
Semáforos, Monitores y Lock

Consideraciones para la realización del práctico:

- Debe tener en cuenta que los valores que se presentan en los enunciados son a modo ejemplo. Deberán probar con distintos valores para asegurar el correcto funcionamiento del programa.
- Recuerde utilizar variables en vez de números fijos.

1. **Problema clásico de concurrencia - Sala de Fumadores.**
   Considere un sistema formado por tres hilos fumadores que se pasan el día armando cigarrillos y fumando. Para armar y fumar un cigarrillo necesitan tres ingredientes: tabaco, papel y fósforos. Cada fumador dispone de un surtido suficiente (para el resto de su vida) de uno de los tres ingredientes. Cada fumador tiene un ingrediente diferente, es decir, un fumador tiene una cantidad infinita de tabaco, el otro de papel y el otro de fósforos. Hay también un hilo agente que pone dos de los tres ingredientes encima de una mesa. El agente dispone de unas reservas infinitas de cada uno de los tres ingredientes y escoge de forma aleatoria cuáles son los ingredientes que pondrá encima de la mesa. Cuando los ha puesto, el fumador que tiene el otro ingrediente puede armar su cigarrillo y fumar (los otros dos no). Para ello toma los ingredientes, se arma un cigarrillo y se lo fuma. Cuando termina de fumar vuelve a repetirse el ciclo. En resumen, el ciclo que debe repetirse es :
   "agente pone ingredientes → fumador hace cigarro → fumador fuma →fumador termina de fumar → agente pone ingredientes → ..."

Es decir, en cada momento a lo sumo hay un fumador fumando un cigarrillo.
Considere el código siguiente e implemente la clase SalaFumadores, como recurso compartido entre fumadores y agente.

```
public class Fumador implements Runnable{
   private int id;
   private SalaFumadores sala;
   public Fumador(int id, SalaFumadores sala){
      this.id = id;
      this.sala = sala;
   }//constructor
   public void run(){
      while(true){
         .....
      try {
         sala.entrafumar(id);
         System.out.println(&quot;Fumador &quot;+id+&quot; está fumando.&quot;);
```

```
        Thread.sleep(1000);
        sala.terminafumar();
      } catch (InterruptedException e) {
        e.printStackTrace();
        }//catch
      }//while
}//run
}// clase
```

```
public class Agente implements Runnable {
  private SalaFumadores sala;
  private Random r;
  public Agente(SalaFumadores sala){
    this.sala = sala;
    r= new Random();
  }
  public void run (){
    while(true){
      sala.colocar(r.nextInt(3)+1);
    }//while
  }// run
}//clase
```

```
public class DisparaSala {

  public static void main(String[] args) {
    SalaFumadores sala = new SalaFumadores();
    Fumador f1 = new Fumador(1, sala);
    Fumador f2 = new Fumador(2, sala);
    Fumador f3 = new Fumador(3, sala);
    Agente ag = new Agente(sala);

    Thread fumador1 = new Thread(f1);
    Thread fumador2 = new Thread(f2);
    Thread fumador3 = new Thread(f3);
    Thread agente = new Thread(ag);

    fumador1.start();
    fumador2.start();
    fumador3.start();
    agente.start();
  }//main
}//clase
```

2. **Sala de museo.**
   Se quiere desarrollar un sistema para controlar la temperatura y el número de personas que se encuentran en una sala de un museo. En condiciones normales, se permiten **50** personas en la sala. Si la temperatura sube por encima de un umbral (**tUmbral** = 30), se limita el número de

personas a **35**. Si cuando se detecta este suceso el número de personas en la sala es mayor que 35, no es necesario desalojarlas. Si una persona jubilada intenta entrar, tendrá prioridad frente al resto de personas que estén esperando. Cada persona se representa mediante un hilo. Además, hay un hilo que mide periódicamente la temperatura de la sala y notifica su valor al sistema. Se pide desarrollar un monitor (*GestorSala*) que sincronice los hilos que representan personas y al hilo que mide la temperatura, de acuerdo con las especificaciones anteriores.

a. Utilizar monitor o locks, proporcionando los siguientes métodos:

```
... void entrarSala()
// se invoca cuando una persona quiere entrar en la sala.

... void entrarSalaJubilado()
// se invoca cuando una persona jubilada quiere entrar en la sala.

... void salirSala()
// se invoca cuando una persona, jubilada o no, quiere salir de la sala.

... void notificarTemperatura(int temperatura)
// lo invoca la hebra que mide la temperatura de la sala para indicar el último valor medido.
```

Observación: No es necesario garantizar que el orden de acceso a la sala coincide con el orden de llegada a la puerta de entrada.

3. **Centro de impresión.**
Retomando el siguiente ejercicio:

> *En un centro de impresión se cuenta con dos tipos distintos de impresoras para dar servicio a los usuarios: impresoras de tipo A e impresoras de tipo B. Obviamente, el número de impresoras de cada tipo es limitado: NumA impresoras de tipo A y NumB impresoras de tipo B. Para imprimir un trabajo en una impresora de un tipo determinado (A o B) es necesario hacer la solicitud indicando el tipo de impresión requerida. A la hora de imprimir los trabajos se pueden considerar tres grupos distintos de procesos usuarios:*
> *Los que requieren una impresora de tipo A.*
> *Los que requieren una impresora de tipo B.*
> *Los que pueden utilizar una impresora de uno cualquiera de los dos tipos.*
> *Los hilos usuarios generan trabajos y los imprimen. Como restricción: dos hilos no pueden ejecutar simultáneamente las operaciones de impresión (Imprimir A o Imprimir B) sobre una misma impresora.*
> *Cuando un usuario quiera imprimir un trabajo deberá hacerlo sobre una impresora compatible con él y que no esté siendo utilizada por otro usuario. En otro caso el proceso deberá esperar.*

a. Considere ahora que hay un buffer de tamaño **kA** compartido por las impresoras de tipo A y un buffer de tamaño **kB** compartido por las impresoras de tipo B.

b. ¿Qué podría suceder si existiera un tercer buffer compartido por todas las impresoras para los trabajos a los que les sirve cualquier tipo de impresora?

4. **Cuartel de soldados.**
   En un cuartel hay un comedor para 100 soldados. Cuando el soldado quiere almorzar, entra en el recinto y toma una bandeja con comida de uno de los 5 mostradores (mostradorAlmuerzo) que existen para tal efecto; la bandeja, además de la comida, tiene un vaso de agua o una botella de gaseosa, si escoge esto último necesita uno de los 10 abridores para poder abrirla. En el caso que el soldado quiera postre, debe dirigirse a uno de los 3 mostradores que lo despachan (mostradorPostre). Cuando el soldado termina de comer, deja la bandeja en la cocina y se va.
   a. Realizar un programa concurrente de forma que utilizando algún mecanismo de sincronización coordine las tareas de los soldados.
   b. Considere ahora que hay menos soldados (por ejemplo 50, 30, 10) y cambia la cantidad de mostradores, en cada ejecución probar con distintos valores. Aplicar la misma idea para los abridores.

5. **Tráfico**.
   Coches que vienen del norte y del sur pretenden cruzar un puente sobre un río. Solo existe un carril sobre dicho puente(de una sola vía, es decir que los coches en la misma dirección no pueden adelantarse). Por lo tanto, en un momento dado, sólo puede ser cruzado por uno o más coches en la misma dirección (pero no en direcciones opuestas).
   a. Implementar la solución con semáforos.
   b. Implementar teniendo en cuenta el código del siguiente **monitor**, que resuelva el problema del acceso al puente.

```
MONITOR GestionaTrafico
    ..............
    PROCEDIMIENTO EntrarCocheDelNorte
    ...................
    PROCEDIMIENTO SalirCocheDelNorte
    ...................
    PROCEDIMIENTO EntrarCocheDelSur
    ...................
    PROCEDIMIENTO SalirCocheDelSur
```

   c. Mejorar la implementación del monitor anterior, de forma que la dirección del tráfico a través del puente cambie cada vez que lo hayan cruzado 10 coches en una dirección, mientras 1 o más coches estuviesen esperando cruzar el puente en dirección opuesta.
   d. Realizar una clase test de prueba a fin de que las implementaciones antes solicitadas puedan ser probadas.

   IMPORTANTE:
   - debe considerar que la implementación asegure que los coches crucen el puente en el orden en que llegaron, es decir, si C1 llegó al puente antes que C2, en la misma dirección, C1 debe poder empezar y terminar de cruzar el puente antes que C2.

- debe tener en cuenta además que

6. **Observatorio.**

Se realiza una visita al observatorio de la ciudad. El mismo permite el ingreso de visitantes, personal de mantenimiento e investigadores.

- los visitantes, que van a estudiar las estrellas,
- las personas de mantenimiento, que se ocupan de controlar que todo esté aseado,
- Los investigadores analizan estrellas y realizan su trabajo de investigación.

La sala tiene capacidad para 50 personas, pero si entra algún visitante en silla de ruedas, la capacidad baja a 30 personas hasta que se retire. Cuando se retira, la capacidad vuelve a ser de 50 personas. Puede ocurrir que el visitante quiera ingresar cuando la capacidad sea mayor a 30, en ese caso debe esperar a que la capacidad disminuya.

Existen cierta restricciones impuestas por el director del observatorio:

- Los visitantes pueden ingresar cuando haya otros visitantes, siempre que no superen la capacidad.
- Las personas de mantenimiento ingresan periódicamente, siempre que no haya visitantes, aunque sí puede haber otras personas de mantenimiento.
- Los investigadores tienen exclusividad, es decir deben estar solos en el recinto ya que se tienen que concentrar para realizar sus observaciones. Las observaciones deben ser guardadas en el libro de observaciones diarias.
- Si un investigador quiere entrar y hay visitantes y/o personal de mantenimiento, debe esperar a que se retiren.

a. Identificar los roles activos y los recursos compartidos en el escenario presentado.
b. Dar una solución utilizando mecanismos de sincronización que modele el comportamiento explicado.
c. Identificar problemas de inanición que puedan surgir y cómo podría solucionarlo.

Nota: Este ejercicio fue tomado en un parcial/recuperatorio.

7. **Pastelería**

Una pastelería tiene tres hornos que producen tres tipos diferentes de pasteles: A, B y C, con pesos diferentes: *pesoA*, *pesoB* y *pesoC*. Procedentes de los hornos, los pasteles se van situando en un mostrador común.

Los pasteles son empaquetados en cajas. Uno o varios robots Empaquetadores toman pasteles del mostrador y los introducen en la caja. Cada caja puede contener un número diferente de pasteles siempre y cuando no se sobrepase un peso límite, denominado *PesoMaximo*. Por este motivo, antes de incluir un pastel en la caja, cada empaquetador debe asegurarse de que con su inclusión no se sobrepasa el peso máximo. Si no se sobrepasa el peso, se incluye el pastel en la caja; en otro caso, un **brazo** mecánico se encarga de retirar la caja que se estaba llenando y posteriormente la sustituye por una caja vacía.

Tener en cuenta de que se trata de llenar cada caja lo más posible, lo cual puede ser conseguido por uno cualquiera de los robots que intentan depositar simultáneamente algún pastel, de pesos que pueden variar. Se considera que no hay interferencia física entre robots que intentan soltar ubicar pasteles al mismo tiempo en la caja.

Se asume que inicialmente hay una caja vacía junto al mostrador.
Considere las siguientes operaciones:

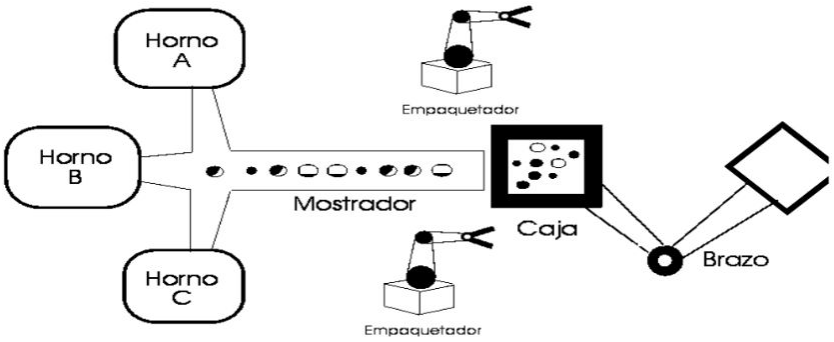
a. Retirar Caja: *hace que el proceso que lo invoca quede bloqueado hasta que la caja que estaba siendo llenada es retirada por el brazo auxiliar . Requiere que haya una caja en la zona de relleno.*

b. Reponer Caja: *hace que el proceso que lo invoca quede bloqueado hasta que el brazo auxiliar coloque una caja vacía en el área de relleno. No debe haber ninguna caja en la zona de rellenado.*

c. Tomar Pastel, retorna Peso: *provoca que el proceso que lo invoca quede bloqueado hasta que el empaquetador toma el pastel más cercano al mostrador, indicando Peso el peso del mismo.*

d. Soltar Pastel: *provoca que el proceso que lo invoca quede bloqueado hasta que el empaquetador suelta el pastel que acaba de tomar en la caja del área de relleno. Es necesario que haya una caja en el área de rellenado, que el robot empaquetador en cuestión tenga un pastel y que la inclusión de ese pastel en la caja no haga sobrepasar el peso máximo permitido. Físicamente, no hay interferencia entre dos robots que intentan depositar pasteles simultáneamente en una misma caja.*

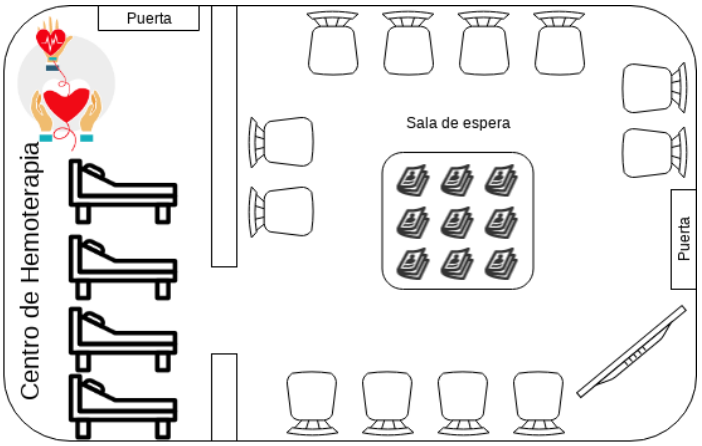
Implementar utilizando los mecanismos de sincronización que crea más conveniente.



8. Centro de hemoterapia.

En un centro de hemoterapia se reciben donaciones de sangre. Para realizar las extracciones, que toman un tiempo aleatorio, se cuenta con 4 camillas. Las personas que llegan a donar sangre son atendidas en orden de llegada, pero mientras no haya camillas libres esperan en la sala de espera, pueden optar por utilizar las sillas o esperar parados.

En la sala de espera hay 9 revistas “Muy Interesante” que las personas leen mientras esperan su turno y un televisor sin audio sintonizado en el canal crónica-tv. Los donantes siempre prefieren leer una revista, pero si no hay ninguna disponible miran el noticiero. Cuando se libera una camilla se llama a la siguiente persona en la sala de espera, quien deja la revista si tenía una. Ni bien una revista es liberada las personas mirando el televisor compiten por tomarla.



a. Identificar los roles activos y los recursos compartidos en el escenario presentado.
b. Dar una solución utilizando mecanismos de sincronización que modele el comportamiento explicado.
   Nota: Este ejercicio fue tomado en un parcial/recuperatorio.