Preguntas del Primer Cuestionario

1.- Para lograr la exclusión mutua de una sección crítica donde se accede a un recurso compartido inicialmente disponible

Seleccione una:
a. los semáforos no sirven para lograr la exclusión mutua
b. si se usa un semáforo binario, debe inicializarse en 1
c. si se usa un semáforo binario, debe inicializarse en 0
d. si se usa un semáforo binario, la inicialización depende del recurso que se comparte

Opcion correcta: b)
Para lograr la exclusión mutua de na sección crítica donde se accede a un recurso compartido inicialmente disponible si se usa un semáforo binario, debe inicializarse en 1.
Ya que si se inicializará en 0 se estaría marcando que el recurso NO esta disponible inicialmente.
Y no depende del recurso que se comparte, ya que cualquiera sea, la exclusión mutua se refiere a que solo un proceso/hilo por vez puede estar ejecutando la sección crítica.

2.- El semáforo elimina la espera activa porque

Seleccione una:
a. el semáforo no elimina la espera activa
b. las operaciones de espera (adquirir) y señal (liberar) se implementan como acciones indivisibles
c. se inicializa al número máximo de recursos que se comparten
d. se implementa con una cola de tareas a la cual se le añaden los procesos que están en espera del recurso

Opción correcta: d)
El semáforo elimina la espera activa porque se implementa con una cola de tareas a la cual se le añaden los procesos que están en espera del recurso, y esos procesos liberan la CPU.

El hecho de que se inicializa al número máximo de recursos que se comparten (o de procesos que tiene permitido actuar de forma concurrente) no tiene relación con la espera activa, es para coordinar el trabajo de esa cantidad de hilos.

3.-
Lista A
a) Las excepciones chequeadas
b) Un bloque finally
c) Si se define un bloque try-catch
d) Una excepción no chequeada (runtime exception)

Lista B
a) se ejecuta sin importar si la excepción fue disparada
b) el bloque finally es opcional

c) deben ser capturadas (catch) o disparadas (throw)
d) normalmente se refiere a un error de programación

Respuestas
- Las excepciones chequeadas deben ser capturadas o disparadas.
Las excepciones chequeadas son las chequeads por el compilados y no se pueden evitar con buena programación, por ejemplo un servidor que esta apagado. El compilador chequea todo menos las "excepciones runtime". El lenguaje obliga a que tales excepciones sean capturadas, es decir consideradas en un bloque "try-catch" con un manejador apropiado, o en caso contrario sean disparadas hacia afuera, es decir declarada con "throws ..." en la declaración del métod, para que otro manejador mas externo la trate o la siga disparando hacia afuera en la cadena de llamadas.

- Un bloque finally se ejecuta sin importar si la excepción fue disparada.
Mientras que el bloque catch solo se ejecuta si la excepción considerada fue disparada.

- Si se define un bloque "try-catch" el bloque finally es opcional. Mientras que si se define un bloque "try" sin "catch", es decir sin un manejador para la excepción, el bloque finally debe estar.
Entonces puedo tener un bloque "try-catch", un bloque "try-finally" o un bloque "try-catch-finally".

- Una excepción no chequeada (runtime exception) normalmente se refiere a un error de programación.
Que la excepción sea no chequeada significa que el compilador no la chequea. Son normalmente errores que pueden evitarse con buena programación (por ejemplo subindice fuera de rango). Hay que tratar de evitarlas. Se producen por problemas en la lógica, NO en una condición que falla en ejecución en forma inesperada (por ejemplo un archivo que no se encuentra). Este tipo de excepciones generalmente se utilizan en la fase de desarrollo y testeo.

4.- Dos procesos serán concurrentes:
Seleccione una:
a. si un proceso ejecuta después de que finalice el otro
b. si existe un solapamiento en la ejecución de sus instrucciones
c. si comparten uno o varios recursos.

Opción correcta: b)
Dos procesos serán concurrentes si existe un solapamiento en la ejecución de sus instrucciones.
Si bien en ocaciones, cuando se tienen varios procesos pareciera que se ejecutan en secuencia, es decir una después que finalice el otro, aun cuando se han programado para ejecutarse de forma concurrente, eso tiene que ver con como el planificador de procesos distribuye y asigna el tiempo de CPU a los procesos.

5.- Suponga la siguiente jerarquía de excepciones creada por usted:
Exception

- excInstrumento
- excViento
- excCuerda
- excPercusión
- ...

y considere el siguiente trozo de código:

```
...
try {
        Musica.hacerMusica()
 } catch (excInstrumento instExc) {
          // tratamiento de la excepcion
 } catch (excViento vieExc) {
          // tratamiento de la excepcion
 }
```

Seleccione una:
a. no es correcto porque falta el finally.
Anteriormente se indico que el bloque finally es opcional.

b. no es correcto porque no se consideran todos los tipos de excepción de la jerarquía.
No es obligación considerar TODAS las subclases que conforman la jerarquía, solo las necesarias en cada caso.

c. no es correcta porque nunca se va a capturar la excepcion excViento
Esta es l aopción correcta. No tiene sentido capturar las excepciones de una jerarquía en el orden de la mas general a la mas específica, ya que cuando se encuentra un manejador apropiado no se sigue con los otros manejadores considerados, luego NUNCA se llegaria a los tipos de excepcion mas específicos como excViento en este caso.

d. es correcto

6.- Los semáforos son componentes pasivos de bajo nivel de abstracción que sirven:
Seleccione una:
a. Para controlar el orden de ingreso a un área compartida.
Si bien los semáforos pueden utilizarse para forzar un orden de ejecución de un conjunto de procesos, deben utilizarse varios semáforos (como en el ejercicio 1 del práctico 3), pero NO se trata del orden de ingreso a un área compartida.

b. Para arbitrar el acceso a un recurso compartido.
Dependiendo de las restricciones respecto al acceso al recurso compartido se puede utilizar un semáforo binario o general.

c. Para arbitrar el acceso a un recurso propio.
Si es un recurso propio, NO es compartido, por lo tanto no hace falta utilizar ningún mecanismo para exclusión mutua ni sincronización de acceso.

7.- Si se escribe un método que puede causar una excepción chequeada (checked exception), su uso debe encerrarse en un bloque try-catch
Seleccione una:
Verdadero
Falso

Es falso por lo que se indico en la respuesta a la pregunta 3. Hay otra opción: la excepción puede declararse en el encabezado del método con una clausula "throws".

8.- La espera activa corresponde a:
Seleccione una:
a. el estado "bloqueado" de un proceso pero no retirado a memoria secundaria

b. la espera que realiza la operación "wait" sobre una variable de condición del monitor
Esto suele llamarse "espera pasiva", dado que el proceso que ejecuta un "wait" libera la CPU, permitiendo que mientras se espera por las condiciones necesarias para avanzar otro proceso pueda progresar en su ejecución.

c. la acción de bloqueo que realiza un semáforo sobre un proceso.
En este caso el proceso libera la CPU, y queda bloqueado por lo que NO esta activo.

d. cuando un proceso se mantiene chequeando una condición y por lo tanto consumiendo ciclos de CPU.
Esto impide que otros procesos puedan utilizar la CPU y asi aprovechar mejor el tiempo de CPU en el ambient econcurrente.

9.- al estado de interbloqueo se llega cuando se dan de manera simultánea las siguientes condiciones
Seleccione una:
a. exclusión mutua- retención y espera - existencia de expropiación - espera circular
b. sección crítica - retención y espera - existencia de expropiación - espera circular
c. exclusión mutua- retención y espera - no existencia de expropiación - espera circular
d. solamente exclusion mutua

El estado de interbloqueo se da cuando 2 o mas procesos quedan bloqueados esperando por un recurso que tiene otro de los procesos del grupo, y todos están en la misma situación, por lo tanto ninguno progresa y luego ninguno logra acceder al recurso que le falta. Es decir, cada proceso tiene la exclusión mutua sobre alguno de los recursos, mientras espera por los otros retiene el recurso, no hay posibilidades de que se le quite el recurso que tiene en su poder (expropiación), y se produce una espera circular (el hilo1 tiene R1 y espera R2, el hilo2 tiene R2 y espera R3, .. el hiloN tiene Rn y espera R1).
Pueden tener mas de un recurso en su poder.

10.- Consider el codigo siguiente:

```
public class DualSynch {

   private Object syncObject = new Object();
   int dato=5;

   public synchronized void f() {
         for(int i = 0; i < 5; i++) {
               dato = dato * 4;
               print("f()" + dato);
               Thread.yield();
         }
```

```
    } //de f

    public void g() {
          synchronized(syncObject) {
               for(int i = 0; i < 5; i++) {
                    dato = dato + 20;
                    print("g()" + dato);
                    Thread.yield();
               }
          }
    } //de g
} // dualSynch


public class SyncObject {
    public static void main(String[] args) {
          final DualSynch ds = new DualSynch();
          Thread hilo = new Thread() {
           public void run() {
                ds.f();
           }
         }
          hilo.start();
          ds.g();
    }// del main
}//de SyncObject
```

Seleccione una o más de una:
a. la salida sienpre es la esperada
b. hay 3 hilos en ejecución
c. el "synchronized (syncObject)" no es para lograr la exclusion mutua sobre el dato compartido
d. hay 2 hilos en ejecución
e. no hay recurso compartido

En esta caso hay 2 opciones correctas.
La opción d), ya que hay 2 hilos en ejecución, el hilo creado referenciado por la variable "hilo" y el hilo de ejecución principal que ejecuta el "main"
La opción c), ya que el synchronized se esta aplicando sobre un objeto particular (syncObject). El syncObject tiene su propio lock implícito que es tomado por el hilo que ejecuta el main, y por otro lado el hilo "hilo" ejecuta el método f() que es sincronizado, luego adquiere el lock implícito del objeto "ds" (instancia de DualSync).
Ambos hilos obtiene locks distintos y ambos trabajan sobre la variable "dato" del objeto compartido ds (que por lo tanto tambien es compartida), sin exclusión mutua.

11.- ¿Cuál de estas transiciones de estado de un proceso/hilo jamas se produce en un sistema normal?
Seleccione una:
a. de listo a bloqueado
b. de activo a bloqueado
c. de activo a listo
d. de bloqueado a listo

La opción correcta es la a). Nunca un proceso/hilo puede pasar de listo a bloqueado ya que la única

opción es que de listo pase a "activo" o "en ejecución".

12.- La exclusion mutua permite:
Seleccione una:
a. El acceso de varios procesos a la vez a un recurso.
b. La ejecución de un bloque de sentencias de forma segura.
c. La implementación de un bloque de sentencias en un solo lenguaje.

La opción correcta es la b)
La exclusión mutua permite la ejecución de un bloque de sestamente lo que se quiere evitarntencias de forma segura.
El acceso de varios procesos a la vez a un recurso es justamente lo que se debe evitar para que no haya estados inconsistentes y condiciones de carrera.

13.- Los programas concurrentes son indeterministas es decir
Seleccione una:
a. pueden arrojar diferentes resultados cuando se ejecutan repetidamente sobre un mismo conjunto de datos de entrada
b. Pueden ejecutarse varias veces y siempre darán el mismo resultado
c. Pueden en una cantidad de ejecuciones determinado dar el mismo resultado

La opción correcta es la a)
Los programas concurrentes son indeterministas, es decir puede arrojar diferentes resultados cuando se ejecutan repetidamente sobre un mismo conjunto de datos de entrada.
Esto es así debido a que cuando están involucrados varios hilos cuyas ejecuciones se entrelazan, pueden darse distintas secuencias de ejecución, todas correctas, dependiendo de como actúe el planificador de procesos, dando y quitando la CPU a los hilos.