Certamen Final 100 minutos. Puede usar Apuntes.

Todas las preguntas tienen igual puntaje.

1.- Haga un programa en C que solicite por teclado una línea y luego la copie en la salida estándar. Se pide que si el usuario se demora más de 15 segundo en escribir la línea, su programa envíe a la salida estándar el mensaje " ... Por favor ingrese ahora una línea:"
Así la ejecución del programa, llamado p1, podría arrojar:
$ p1
Ingrese una línea: /* luego de más de 15 segundos */
... Por favor ingrese ahora una línea:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char* argv )
{
  fd_set readfds, readfdsCopy;
  int n,i;
  struct timeval timeout;
  char line[80];

  printf("Ingrese una linea:"); fflush(stdout);
  FD_ZERO(&readfdsCopy);
  FD_SET(0,&readfdsCopy); /*0 es stdin */

  for(;;){
    memcpy(&readfds, &readfdsCopy, sizeof(fd_set));
    timeout.tv_sec=15;  /* 15 segundos*/
    timeout.tv_usec=0;
    n = select(FD_SETSIZE, &readfds, (fd_set *) 0, (fd_set *) 0, &timeout);
    if (n > 0) {
      if (FD_ISSET(0, &readfds)) {
        read (STDIN_FILENO, line, sizeof(line));
        printf("%s", line); /* copia la linea */
        exit(0);
      }
    }else   /* timeout!!*/
      printf("\n ...Por favor ingrese ahora una linea:"); fflush(stdout);
  }
}
```

2.- Considere el siguiente programa:

```
public class Demo {
    synchronized void a() { actBusy(); }

    static synchronized void b() { actBusy(); }

    static void actBusy() {
        try {
            Thread.sleep(1000);
        }catch (InterruptedException e) {}
    }
```

```
    public static void main(String[] args) {
        Demo x = new Demo();
        Demo z = new Demo();
        Runnable runnable = new myRunnable(x,z);
        Thread thread1 = new Thread(runnable);
        Thread thread2 = new Thread(runnable);
        thread1.start();
        thread2.start();
    }
}
class myRunnable implements Runnable {
    Demo x,z;
    public myRunnable(Demo x, Demo z){
        this.x=x;
        this.z=z;
    }
    public void run() {
        int option = (int) (Math.random() * 4);
        switch (option) {
            case 0: x.a(); break;
            case 1: x.b(); break;
            case 2: z.a(); break;
            case 3: z.b(); break;
        }
    }
}
```

¿Cuáles de las siguientes invocaciones de métodos nunca podrían ser ejecutadas concurrentemente? Señale todos los casos que apliquen. Justifique brevemente aquellas que no puedan ocurrir.

A. x.a() en thread1, y x.a() en thread2

B. x.a() en thread1, y x.b() en thread2

C. x.a() en thread1, y z.a() en thread2

D. x.a() en thread1, y z.b() en thread2

E. x.b() en thread1, y x.a() en thread2

F. x.b() en thread1, y x.b() en thread2

G. x.b() en thread1, y z.a() en thread2

H. x.b() en thread1, y z.b() en thread2

Los objetos x y z posee cada uno un candado. Adicionalmente la clase Demo posee su propio candado para controlar acceso a métodos sincronizados de la clase. Ambos hilos comparten los objetos x y z, por lo tanto:

A: No pueden ocurrir concurrencia en estos llamados, pues el candado de x lo impide.

B: Sí puede ocurrir, pues los llamados afectan a distintos candados.

C: Sí puede ocurrir pues cada llamado a fecta al candado del propio objeto.

D: Sí puede ocurrir, los cadados distintos el de un objeto y la clase.

E: Sí puede ocurrir, los candados son el de la clase y el de x.

F: No puede ocurrir, lo impide el candado de la clase.

G:Sí puede ocurrir, son candados distintos.

H: No puede ocurrir, lo controla el candado de la clase.

3.- Se desea que el programa adjunto muestre por pantalla el valor del atributo **a** impreso por cada hilo en forma alternada (no dos veces seguida un mismo hilo). Señale si el programa adjunto obtiene lo deseado. En caso contrario, haga una modificación del programa para lograr lo deseado.

Resultado deseado	Resultado no deseado
$ java Hilo 0 0 1 1 : /* sigue*/	$ java Hilo 0 1 2 0 1 :

```
import java.lang.*;
public class Hilo extends Thread {
   private int a=0;
   public static void main(String[] args) {
      Hilo h1 = new Hilo();
      Hilo h2 = new Hilo();
      h1.start();
      h2.start();
   }
   public void run () {
      while(true) {
        try{
        System.out.println(a);
         a++;
         sleep((int)(100*Math.random()));
         }catch (InterruptedException e) {}
      }
   }
}
```

El programa mostrado no arroja lo deseado.

```
import java.lang.*;
public class Hilo extends Thread {
  private int a=0;
  private Hilo otro;  /* para que el hilo pueda ceder el turno */
  private Object candado; /* cada hilo referencia un único candado*/
  static Hilo turno;  /* común para ambos hilos para informar turno*/
  public static void main(String[] args) {
    Hilo h1 = new Hilo();
    Hilo h2 = new Hilo();
    Object cualquiera = h1;
    h1.setOtroYcandado(h2, cualquiera);
    h2.setOtroYcandado(h1, cualquiera);
    turno=h2;
    h1.start();
    h2.start();
  }
  public void setOtroYcandado(Hilo o, Object c) {
     otro=o;
     candado=c;
```

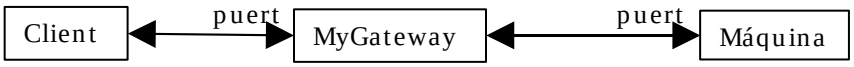
```
    }
  public void run () {
    while(true) {
      try{
        synchronized(candado) {
          if (turno!=this)
            try { candado.wait();
            } catch(InterruptedException e) {       }
          turno=otro;
          candado.notify();
        }
        System.out.println(a);
        a++;
        sleep((int)(1000*Math.random()));
      }catch (InterruptedException e) {}
    }
  }
}
```

Otra solución para el problema es:

```
import java.lang.*;
public class Hilo2 extends Thread {
  private int a=0;
  private Object candado;
  public static void main(String[] args) {
    Hilo2 h1 = new Hilo2();
    Hilo2 h2 = new Hilo2();
    Object cualquiera =h1;
    h1.setCandado(cualquiera);
    h2.setCandado(cualquiera);
    h1.start();
    h2.start();
  }
  public void setCandado(Object c) {
    candado=c;
  }
  public void run () {
    while(true) {
      try{
        synchronized(candado){
          System.out.println(a);
          a++;
          candado.notify();
          candado.wait();
        }
        sleep((int)(1000*Math.random()));
      }catch (InterruptedException e) {}
    }
  }
}
```

4.- Pasarela o Gateway en Java: Se pide desarrollar un programa Java que redirija conexiones TCP hacia una máquina dada. Para invocar tal programa usamos:
$ java myGateway <máquina> <puerto>
myGateway espera en el puerto dado conexiones de clientes. Cuando una conexión llega, myGateway establece una conexión con ese mismo puerto en la máquina dada. Luego

myGateway envía a <máquina> todo lo que llegue desde el cliente. Del mismo modo myGateway envía hacia el cliente todo lo que llegue desde <máquina>.
Obs: myGateway atiende a un cliente a la vez. Asuma que la comunicación incluye sólo líneas de texto.




```
import java.io.*;
import java.net.*;

public class MyGateway
{
  public static void main(String[] args ) {
    try   {
        // establish server socket
      ServerSocket s = new ServerSocket(Integer.parseInt(args[1]));
      while(true) {   // wait for client connection
        Socket clientSide_s = s.accept( );
        Socket serverSide_s = new Socket(args[0],Integer.parseInt(args[1]));
        Tunel client2server_t= new Tunel(clientSide_s, serverSide_s);
        Tunel server2client_t= new Tunel(serverSide_s, clientSide_s);
        client2server_t.start();
        server2client_t.start();
        client2server_t.join();
        server2client_t.join();
        clientSide_s.close();
        serverSide_s.close();
      }
    }  catch (Exception e){ }
  }
}

class Tunel extends Thread
{
  private Socket incoming, outgoing;
  public Tunel (Socket in_s, Socket out_s)    {
      incoming=in_s;
      outgoing=out_s;
  }
  public void run()    {
      try    {
        BufferedReader in = new BufferedReader
          (new InputStreamReader(incoming.getInputStream()));
        PrintWriter out = new PrintWriter
          (outgoing.getOutputStream(), true /* autoFlush */);
        boolean done = false;
        while (!done)    {
          String line = in.readLine();
          if (line == null) done = true;
          else
            out.println(line);
        }
      }    catch (Exception e)    {
        e.printStackTrace();
      }
    }
}
```