PRACTICA

PROGRAMACION CONCURRENTE

CURSO 2005-2006

NOMBRE :
VICTOR M. DEL CANTO GODINO

DIRECCION:
C/ HIGINIO MANGAS 10 4-B 47005
VALLADOLID

DNI:
12759808Y

TELEFONO:
983211989

CORREO ELECTRONICO:
VCANTO1@ALUMNO.UNED.ES

INGENIERÍA TÉCNICA en INFORMÁTICA de SISTEMAS
PROGRAMACIÓN CONCURRENTE
PRACTICA OBLIGATORIA CURSO 2005-2006

1. Enunciado

Un edificio de 16 plantas tiene un único ascensor. Cuando nadie utiliza el ascensor, este está parado. Cuando una persona quiere subir al ascensor pulsa el botón de llamada desde la planta en la que se encuentra. El ascensor tiene en cuenta dichas llamadas. Cuando la gente está dentro del ascensor pulsa el botón del piso en el quiere bajar. En el ascensor caben 8 personas simultáneamente. El ascensor tiene memoria, de modo que pueden solicitarse varias subidas y bajadas al mismo tiempo y el ascensor las irá atendiendo cuando sea posible. El ascensor tiene una política de no cambiar de sentido si no es estrictamente necesario. Cuando va bajando atenderá primero las peticiones que impliquen desplazarse hacia abajo hasta que ya no tenga más peticiones pendientes más abajo y hará lo propio en sentido ascendente. Programar en pseudocódigo el comportamiento del ascensor utilizando dos tipos de proceso: Persona y Ascensor. La eficiencia es importante. Para seguir lo que está ocurriendo en cada momento es menester que tanto el ascensor como las personas emitan mensajes por pantalla de vez en cuando para saber qué es lo que está ocurriendo, por ejemplo: 'Persona 9: Pulso el botón para subir al piso 14', 'Persona 4: El ascensor está lleno, no puedo subir', 'Ascensor: Voy bajando por el piso 3', 'Ascensor: Me detengo para que baje un viajero en planta 5', etc...

2. Trabajo del alumno

Se pide:

- Discutir las estructuras de datos necesarias para programar los procesos de Ascensor y Persona.
- Analizar qué partes del código deben ejecutarse en exclusión mutua y cuáles pueden y deben ejecutarse concurrentemente.
- Discutir las condiciones de sincronización entre los procesos.
- Discutir las ventajas e inconvenientes de cada una de las herramientas de comunicación y sincronización entre procesos que se presentan a continuación de cara a la programación de este problema concreto y decidir si son adecuadas o no

    1. Variables compartidas y semáforos

2. Regiones críticas condicionales
3. Monitores
4. Buzones
5. Canales
6. Rendez-vous

- Elegir dos opciones de la lista de alternativas anterior y programar el enunciado en pseudocódigo. Es muy importante no mezclar primitivas (por ejemplo, monitores con semáforos y RCC).
- Elegir DOS lenguajes de programación concurrente (PASCAL-FC, JAVA, Ada, C...) e implementar cada una de las soluciones propuestas en uno distinto. Si el lenguaje no dispone de las herramientas elegidas, simularlas mediante las primitivas del lenguaje en cuestión (ejemplo, en JAVA no hay semáforos pero son fáciles de simular).

Es muy importante respetar el orden de realización de los apartados. No sirve de nada hacer los dos programas si antes no se ha razonado cuales son las mejores opciones y se ha analizado la concurrencia y sincronización antes de escribir el pseudo-código.

Las prácticas serán entregadas por correo ordinario o siguiendo las instrucciones que se publicarán en su momento en la página web. Nunca por correo electrónico.

La dirección de entrega de la práctica es la siguiente:

David Fernández Amorós
Despacho 2.16
ETSI Informática UNED
C/Juan del Rosal 16
28040 Madrid

Para poder aprobar la asignatura en una convocatoria determinada, la práctica debe estar entregada antes del comienzo del periodo de exámenes de dicha convocatoria.

Cualquier duda con respecto a la práctica puede plantearse en el curso virtual de la asignatura, durante las guardias o por correo electrónico escribiendo a programacion.concurrente@lsi.uned.es.

A pesar de lo indicado en la guía del curso, la práctica es de carácter obligatorio, no obstante, su realización puede aumentar la nota final de la asignatura hasta en un punto.

1. Discutir las estructuras de datos necesarias para programar los procesos de Ascensor y Persona.

Tenemos dos procesos: Ascensor y Persona. Cada Persona envía una orden al Ascensor que puede ser de tres tipos:

- Llamada, cuando la Persona esta fuera y desea hacer uso del ascensor
- Subir, cuando esta dentro y desea subir a un piso de mas altura que el de origen
- Bajar, cuando esta dentro y desea bajar a un piso de menos altura que el de origen

Los datos que cada proceso Persona maneja son el piso de origen y el piso destino. Por tanto tenemos un par de valores <operación, piso>.
Operación puede ser llamada, subir o bajar (estos últimos se obtienen del signo de la diferencia entre el piso de origen y destino). El piso según el enunciado esta comprendido en un rango de enteros de 0 a 15, 16 plantas.

Asumo que el número de llamadas que se pueden hacer desde cada planta es único, independientemente del número de personas que se encuentran en dicha planta. Así que por simplicidad, en principio, establezco la regla de que solo una persona estará disponible en cada planta como máximo. A este tema volveré mas adelante.

La estructura que usaré, será un array bidimensional (0..2) (0..15), en la que el primer valor indica el tipo de operación (0-llamada, 1-subir, 2-bajar), y el segundo valor indica el piso. Si la operación es llamada el piso será el de origen, si es subir o bajar el piso será el de destino.

Esto implica que dos personas no puedan subir o bajar al mismo piso aunque vengan de orígenes distintos. Lo que supone una limitación importante, por lo que en uno de los programas, y en lenguaje que lo permite como JAVA, uso estructuras mas complejas, que viene implementadas como la lista enlazada.
Esta estructura si que permite que dos Personas con distinto origen puedan ir al mismo destino puesto que este no esta indicado en el índice del array, sino que es un elemento de una lista ordenada, y enlazada. También permite tener más de una llamada desde el mismo origen, pero supone más complejidad en el código puesto que se necesita un iterador para recorrer la lista. En PASCALFC esta estructura, la lista enlazada, no esta implementada de por si.

Como comprobé después esta limitación resulto exagerada, puesto que la exclusión mutua entre los procesos evita que se produzca un caso de riesgo, como pudiera ser el borrado de una llamada por haberse atendido la llamada de otra Persona desde el mismo origen, por ejemplo.

El hecho de usar un array es mas sencillo pero tiene un handicap. Según el enunciado, el Ascensor tiene una política de no cambiar de sentido si no es necesario, lo cual hace que existan situaciones conflictivas, por ejemplo si en cierto momento el ascensor se queda vacío y existen llamadas en pisos inferiores y superiores, o, como saber cual esta mas cercano, el destino de una Persona que esta en el interior o una llamada de una Persona del exterior.
Con un array tenemos que recorrerlo hasta encontrar el índice adecuado, en cambio con la lista enlazada y ordenada esto nos viene directamente en el siguiente elemento. Por eso

tuve que crear unas funciones (minsuperior y mininferior) en PASCALFC que resolvieran este tipo de problemas.

Además de estas estructuras usadas he de mencionar otra que tuve que utilizar en JAVA para el paso de mensajes para asociar cada proceso Persona con su piso origen y su canal. Uso un TreeMap cuya clave es el piso de origen y el valor es otro TreeMap con clave Persona y valor el canal. Esto me permite que en un piso determinado a cierta Persona se le envíe un mensaje por el canal adecuado para realizar la acción que corresponda.

2. Analizar qué partes del código deben ejecutarse en exclusión mutua y cuales pueden y deben ejecutarse concurrentemente.

Tenemos los siguientes datos:

Nº de plantas=16 (constante)

Proceso Ascensor

Variables
Subida – sentido anterior del ascensor
Interior – numero de personas en el interior
Pisoactual – piso en que se encuentra actualmente

Acciones
Ir – movimiento a realizar de subida o bajada

Proceso Persona

Variables
Origen – piso de origen
Destino – piso destino

Acciones
Llamar – llamada desde el piso de origen
Entrar – entra en el ascensor
Salir – sale del ascensor

Existe una estructura de datos compartida por los procesos Personas y el proceso Ascensor independientemente del lenguaje elegido y la herramienta de sincronización, compuesta básicamente de dos tipos de datos, el piso de origen y el de destino.
El acceso a esta estructura se realizara en exclusión mutua. Los procesos Personas acceden a esta estructura para realizar la acción Llamar que implica una inserción de sus variables en la estructura. El proceso Ascensor necesita consultar la estructura para determinar la variable Subida y la acción Ir, pero mientras se realiza no debe haber nuevas inserciones.
En cuanto a como determinar la acción Ir y la variable Subida, hay que tener en cuenta la política del ascensor. Esto puede resumirse así:

- El ascensor esta vacío(Interior=0)
  - No existen llamadas NADA

- Existen solo llamadas de pisos inferiores al actual BAJAR
- Existen solo llamadas de pisos superiores al actual SUBIR
- Existen llamadas de pisos inferiores y superiores al actual
  - Actual – Inferior > Superior – Actual SUBIR
  - Actual – Inferior < Superior – Actual BAJAR
  - Actual – Inferior = Superior – Actual
    SUBIR o BAJAR depende del sentido anterior a estar vacío

- El ascensor esta lleno (Interior=8)
  - Bajaba(Subida=0) y hay personas dentro que bajan BAJAR
  - Bajaba(Subida=0) y no hay personas dentro que bajan SUBIR
  - Subía(Subida=1) y hay personas dentro que suben SUBIR
  - Subía(Subida=1) y no hay personas dentro que suben BAJAR

- El ascensor esta mediado(0<Interior<8)
  - Bajaba(Subida=0) y existen llamadas de pisos inferiores al actual BAJAR
  - Bajaba(Subida=0) y hay personas dentro que bajan BAJAR
  - Subía(Subida=1) y existen llamadas de pisos superiores al actual SUBIR
  - Subía(Subida=1) y hay personas dentro que suben SUBIR

3. Discutir las condiciones de sincronización entre los procesos.

Proceso Ascensor

Variables
Subida – sentido anterior del ascensor
Interior – numero de personas en el interior
Pisoactual – piso en que se encuentra actualmente

Proceso Persona

Variables
Origen – piso de origen
Destino – piso destino

Acciones
Entrar – entra en el ascensor: si Pisoactual = Origen y Interior<8
Salir – sale del ascensor: si Pisoactual = Destino

4. Discutir las ventajas e inconvenientes de cada una de las herramientas de comunicación y sincronización entre procesos que se presentan a continuación de cara a la programación de este problema concreto y decidir si son adecuadas o no

1. Variables compartidas y semáforos
2. Regiones críticas condicionales
3. Monitores
4. Buzones
5. Canales
6. Rendez-vous

Dentro del grupo de soluciones basadas en variables compartidas la elección que hice es usar un monitor, que proporcionan más estructuración que las RCC y pueden ser implementados eficientemente como los semáforos. Los semáforos podrían ser una buena solución y de hecho uso alguno pero tienen la desventaja de tener que controlar su posición en el código y la complejidad que producen (hay que usar unos cuantos). La primera opción que use fue una Región Critica Condicional, pero desistí de usarla por no estar implementada en el lenguaje elegido y eso me hizo decidirme por el monitor. Los procedimientos del monitor son: Arranque, Parada (ambos para permitir el transito entrada/salida), Iniciar (inicializar el array), Llamada (insertar en el array), InicioConsulta, FinConsulta (ambos para bloquear el array mientras se consulta los datos), Entrar, Salir (con condición de sincronización).

Para el grupo de soluciones basadas en el paso de mensajes me plantee el problema desde el punto de vista de los procesos Personas que son los procesos emisores.

…
llamar
espera (llegada del ascensor no lleno)
entrar
espera (desplazamiento hasta el piso destino)
salir
..

La persona envía un mensaje con los datos y espera a la respuesta para entrar. Una vez dentro envía un mensaje y espera la respuesta para salir.

Esto no se corresponde con un esquema de buzones sino con uno sincrono, pero se corresponde claramente con una invocación remota (extended rendez-vous), donde el proceso emisor (Persona) queda bloqueado esperando por una respuesta del proceso receptor (Ascensor).

5. Elegir dos opciones de la lista de alternativas anterior y programar el enunciado en pseudocódigo. Es muy importante no mezclar primitivas (por ejemplo, monitores con semáforos y RCC).

MONITOR

Monitor

Procedimientos
Llamada, InicioConsulta, FinConsulta, Entrar, Salir.

Proceso Persona

Llamada (origen, destino)
Entrar
Salir

Proceso Ascensor

Procedimiento calculoPisoVacio
InicioConsulta
Calcular el piso según política explicada anteriormente
FinConsulta
Devuelve valorpiso

Procedimiento calculoPisoLLeno
InicioConsulta
Calcular el piso según política explicada anteriormente
FinConsulta
Devuelve valorpiso

Procedimiento calculoPisoMediado
InicioConsulta
Calcular el piso según política explicada anteriormente
FinConsulta
Devuelve valorpiso

Si Ascensor = vacío entonces Ir (calculoPisoVacio)
Si no y Ascensor = lleno entonces Ir (calculoPisoLleno)
Si no y Ascensor = mediado entonces Ir (calculoPisoMediado)

INVOCACION REMOTA

Proceso Persona

Enviar mensaje Llamada (origen, destino)
Si respuesta Entrar
Enviar mensaje Entrada
Si respuesta Salir

Proceso Ascensor

Repetir
  Recibir mensaje Llamada
  Insertar datos en listaLlamada enlazada y ordenarla
  Insertar canalrespuesta y persona en mapa ordenado en árbol
  Tratamiento de datos.
    Si Ascensor = vacío entonces Ir (calculoPisoVacio según política explicada)
    Si no y Ascensor = lleno entonces Ir (calculoPisoLleno según política explicada)
    Si no y Ascensor = mediado entonces Ir (calculoPisoMediado según política explicada)
  Enviar mensaje a personas por su canalrespuesta que estén en pisoactual incrementando contador interior.
  Recibir mensaje Entrada
  Insertar datos en listaSubida enlazada o listaBajada enlazada y ordenarlas
  Insertar canalrespuesta y persona en mapa ordenado en árbol
  Enviar mensaje a personas por su canalrespuesta que estén en pisoactual decrementando contador interior.

6. Elegir DOS lenguajes de programación concurrente (PASCAL-FC, JAVA, Ada, C. . .) e implementar cada una de las soluciones propuestas en uno distinto. Si el lenguaje no dispone de las herramientas elegidas, simularlas mediante las primitivas del lenguaje en cuestión (ejemplo, en JAVA no hay semáforos pero son fáciles de simular).

MONITOR EN PASCAL FC

```
program practicav1;

const
Nper=15;

type
(* llamada es 0,subir es 1,bajar es 2 *)
claselista=array [0..2,0..15] of integer;

var
pisoactual: integer;
subir:integer;
npers:integer;
lista:claselista;


monitor cabina;
export
arranque,parada,llamada,iniciar,entrar,salir,inicioconsulta,finconsulta;
var
actual : array[0..15] of condition;
```

```
sem:boolean;
csem:condition; (*Semaforo para la consulta de datos *)
para:condition ; (*Detiene el ascensor para el transito de Personas*)


procedure insertar (origen,destino:integer;i:integer);
var j:integer;
begin
lista[i,origen]:=destino;
end;


procedure eliminar (celda:integer;i:integer);

begin

lista[i,celda]:=-1;

end;


procedure inicioconsulta;
begin
if sem then delay(csem);
sem:=true;
end;


procedure finconsulta;
begin
if not empty(csem) then resume(csem)
else sem:=false;
end;

procedure entrar(origen,destino,subida:integer);

begin

repeat
if (pisoactual<>origen) then delay(actual[origen])
else
if ((pisoactual=origen)and(npers>7))then
begin
writeln('persona: ',origen:2,'no puedo subir al ascensor esta lleno.');
writeln('persona: ',origen:2,' BLOQUEADA  El ascensor esta en el piso
',pisoactual:2);
delay(actual[origen]);
end;
resume (actual[origen]);
until ((pisoactual=origen)and(npers<8));

if subida>0 then insertar (destino,destino,1)
else insertar (destino,destino,2);
eliminar (origen,0);
npers:=npers+1;
writeln('persona ',origen:2,':entra en el ascensor en el piso
',pisoactual:2);
writeln;

if not empty(actual[origen]) then resume(actual[origen])
end;
```

```
procedure salir(origen,destino,subida:integer);

begin
if pisoactual<>destino then delay(actual[destino]);
if subida>0 then eliminar (destino,1)
else eliminar (destino,2);
npers:=npers-1;
writeln('persona ',origen:2,':sale del ascensor en el piso
',pisoactual:2);
writeln;
if not empty(actual[destino])then resume(actual[destino]);
end;


procedure llamada(origen,destino:integer);

begin
insertar (origen,destino,0);
writeln('persona ',origen:2,': llama para ir al piso ',destino:2);
writeln;
end;


procedure parada;

begin
if not empty(actual[pisoactual]) then
begin
resume(actual[pisoactual]);
delay(para);
end;

end;


procedure arranque;

begin
resume(para);
end;


procedure iniciar;
var i,j:integer;
begin
for i:=0 to 2 do
for j:=0 to 15 do
lista [i,j]:=-1;
end;


begin (*cuerpo de cabina *)
sem:=false;
npers:=0;
subir:=1;
pisoactual:=0;
end; (*fin del cuerpo de cabina*)
```

```
process type persona (origen,destino:integer);

var
subida,sube:real;
sub:integer;
begin

subida:=(destino-origen)/(ABS(destino-origen)+1);
if (subida<>0.0) then
begin
sub:=trunc(subida/abs(subida));
sube:=((sub+1)/2);
sub:=trunc(sube);
cabina.llamada(origen,destino);
cabina.entrar(origen,destino,sub);
cabina.arranque;
cabina.salir(origen,destino,sub);
cabina.arranque;
end;
end;



process type ascensor;


function vacio (i:integer):BOOLEAN; (*devuelve cierto si la lista esta
vacia*)

var n:integer;
vacia:boolean;
begin

vacia:=true;
for n:=0 to 15 do
if (lista[i,n]<>-1) then vacia:=(vacia and false)
else vacia:=(vacia and true);
vacio:=vacia;
end;


function minsuperior (ahora:integer;i:integer):integer;

var maximo,n,j:integer;

begin

maximo:=0;
n:=ahora+1;

while ((n<16) and (maximo=0)) do
begin
if (lista[i,n]<>-1) then maximo:=n;
n:=n+1;
end;
minsuperior:= maximo;
end;
```

```
function mininferior (ahora:integer;i:integer):integer;

var minimo,n,j:integer;


begin

minimo:=16;
for n:=0 to ahora-1 do
begin
if (lista[i,n]<>-1) then minimo:= n;
end;
mininferior:=minimo;
end;



procedure ir(adonde:integer);

begin
if pisoactual<adonde then
begin
subir:=1;
writeln('Ascensor:voy subiendo por el piso ',pisoactual:2);
writeln;
end
else
begin
subir:=0;
writeln('Ascensor:voy bajando por el piso ',pisoactual:2);
writeln;
end;
pisoactual:=adonde;
writeln('Ascensor:me detengo para la entrada/salida de personas en la
planta',pisoactual:2);
writeln;

end;



procedure nadie ;

var cercano: integer;
begin
cabina.inicioconsulta;
cercano:= abs(pisoactual-minsuperior(pisoactual,0))-abs(pisoactual-
mininferior(pisoactual,(*lista,*)0));

if ((cercano< 0) or (cercano=0)) then subir:=1;

if (cercano> 0) then subir:=0;
cabina.finconsulta;
if (subir=1) then
begin
ir(minsuperior(pisoactual,0));
sleep(1);
end
else
begin
```

```
ir(mininferior(pisoactual,0));
sleep(1);
end
end;



procedure lleno ;

begin
cabina.inicioconsulta;
if (vacio(2)=true) then subir:=1;

if (vacio(1)=true) then subir:=0;
cabina.finconsulta;
if (subir=1) then
begin
ir(minsuperior(pisoactual,1));
sleep(1);
end
else
begin
ir(mininferior(pisoactual,2));
sleep(1);
end
end;



procedure mediado ;


begin
cabina.inicioconsulta;
if ((subir=0) and ((mininferior(pisoactual,0)<>16) or (vacio(2)=false)))
then subir:=0

else if ((subir=1) and ((minsuperior(pisoactual,0)<>0) or
(vacio(1)=false))) then subir:=1

else subir:= ((-1)*subir)+1;  (*inversion del sentido*)
cabina.finconsulta;
if (subir=0) then
begin
if (mininferior(pisoactual,0)=16) then ir(mininferior(pisoactual,2))
else if (vacio(2)=true) then ir(mininferior(pisoactual,0))
else if (mininferior(pisoactual,0)<mininferior(pisoactual,2)) then
ir(mininferior(pisoactual,2))
else ir(mininferior(pisoactual,0));
end;

if (subir=1) then
begin
if (minsuperior(pisoactual,0)=0) then ir(minsuperior(pisoactual,1))
else if (vacio(1)=true) then ir(minsuperior(pisoactual,0))
else if (minsuperior(pisoactual,0)>minsuperior(pisoactual,1)) then
ir(minsuperior(pisoactual,1))
else ir(minsuperior(pisoactual,0));
end;

end;
```

```
var j:integer;

begin

repeat
if ((npers=0) and (vacio(0)=true)) then null
else if ((npers=0) and (vacio(0)=false)) then nadie
else if (npers=8) then lleno
else mediado;
cabina.parada;
forever

end;

var
i:integer;
elevador:ascensor;
pasajero:array[0..Nper]of persona;

begin
cabina.iniciar;
cobegin

elevador;

pasajero[0](0,12);
pasajero[1](1,0);
pasajero[2](2,0);
pasajero[3](3,12);
pasajero[4](0,12);
pasajero[5](5,15);
pasajero[6](6,11);
pasajero[7](7,11);
pasajero[8](0,12);
pasajero[9](9,6);
pasajero[10](10,2);
pasajero[11](11,4);
pasajero[12](11,13);
pasajero[13](13,12);
pasajero[14](14,14);
pasajero[15](15,5);
coend

end.
```

INVOCACION REMOTA EN JAVA

SELECT.JAVA

```
package ascensor;
import java.util.*;

public class Select {
      Vector lista=new Vector();

      public void añadirCanal(Seleccionable s){
```

```
            lista.addElement(s);
            s.setSelect(this);
      }

      private void cerrarTodos(){
            for (Enumeration e=lista.elements();e.hasMoreElements();){
                  ((Seleccionable)e.nextElement()).setAbierto(false);
            }
      }

      private void abrirTodos(){
            for (Enumeration e=lista.elements();e.hasMoreElements();){
                  Seleccionable s=(Seleccionable)e.nextElement();
                  if (s.testGuarda())s.setAbierto(true);
            }
      }

      private int testTodos(){
            int i=0;
            int j=1;
            for (Enumeration
e=lista.elements();e.hasMoreElements()&&i==0;++j){
                  Seleccionable s=(Seleccionable)e.nextElement();
                  if (s.hayListos()&&s.testGuarda()){
                        i=j;
                  }
            }
            return i;
      }

      public synchronized int elegir(){
            int indiceSeleccionado=0;
            while (indiceSeleccionado==0){
                  indiceSeleccionado=testTodos();
                  if (indiceSeleccionado==0){
                        abrirTodos();
                        try {wait();}
                        catch (Exception e){}
                        cerrarTodos();
                  }
            }
      return indiceSeleccionado;
      }

}
```

SELECCIONABLE.JAVA

```
package ascensor;

public class Seleccionable {

      private boolean abierto=false;
      private int listos=0;
      private Select select=null;
      private boolean guarda=true;

      void setSelect (Select s) {
            select=s;
      }
```

```
	synchronized void block() {
		if (select==null){
			setAbierto(true);
			while (hayListos()==false)
				try{
					wait();
				}
			catch (Exception e){}
			setAbierto(false);
		}
	}

	synchronized void signal (){
		if (select==null) {
			añadirEmisor();
			if (getAbierto())
				notifyAll();
			}
		else {
			synchronized (select){
				añadirEmisor();
				if (abierto) select.notifyAll();
			}
		}
	}

	boolean hayListos() {
		return listos>0;
	}

	void añadirEmisor(){
		    listos++;
	}

	void añadirReceptor(){
		listos--;
	}

	void setAbierto (boolean v){
		abierto=v;
	}

	boolean getAbierto(){
		return abierto;
	}

	public void asignarGuarda (boolean g){
		guarda=g;
	}

	boolean testGuarda (){
		return guarda;
	}
}
```

CANAL.JAVA

```
package ascensor;

public class Canal extends Seleccionable{
```

```
      Object canal=null;

      public Canal(){
      }

      public synchronized void send (Object o){
                canal=o;
            signal();
                while (canal!=null)
                  try{wait();}
                  catch (Exception e){}
      }

      public synchronized Object receive(){
                block();
            añadirReceptor();
            Object temp=canal;
            canal=null;
            notifyAll();
            return temp;
      }

}
```

PUERTO.JAVA

```
package ascensor;

import java.util.*;

public class Puerto extends Seleccionable{
    Vector cola=new Vector();

    public Puerto() {
    }

    public synchronized void send (Object o) {
        cola.addElement(o);
        signal();
    }

    public synchronized Object receive() {
        block();
        añadirReceptor();
        Object temp= cola.firstElement();
        cola.removeElementAt(0);
        return temp;
    }

}
```

ENTRY.JAVA

```
package ascensor;
import java.util.*;
import java.lang.*;

public class Entry extends Puerto{
    public Mensaje m;
```

```
    public Canal canalCliente=new Canal(); //el canal donde espera la
respuesta

    public Entry() {
    }

    public Object call(Object peticion,int persona) {
        Canal canalCliente=new Canal(); //el canal donde espera la
respuesta
        send (new Mensaje(peticion,canalCliente,persona)); //send por el
puerto
        return canalCliente.receive(); //espera por la respuesta receive
por el canal
    }

    public Mensaje accept() {
        m=(Mensaje)receive(); //receive por el puerto
        return m;
    }

    public void reply(Object res,Canal canalRespuesta) {
        canalRespuesta.send(res);//send por el canal
    }

    public Object answer() {
        return canalCliente.receive();
    }

    public class Mensaje {
        Object peticion;
        Canal canalRespuesta;
        int persona;

        Mensaje(Object m, Canal c,int p) {
            peticion=m;
            canalRespuesta=c;
            persona=p;
        }
    }
}
```

PERSONA.JAVA

```
package ascensor;

import ascensor.Cabina.*;
import java.lang.*;
import java.util.*;

public class Persona extends Thread {
      int sub,origen,destino;
        long espera;
      static short subida,sube;
      static boolean activo=false;
      private static int npersona=0;
      private int conteopersona=++npersona;
      private static Entry llamar,llamada,entrada;
      public Persona(Entry llama,Entry entra,int or,int des) {
                origen=or;
                destino=des;
                llamada=llama;
```

```
                entrada=entra;
            activo=true;
      }

      public void run() {
            while(true){
                  /*try {*/
                      if ((destino-origen)!=0){
                          }else {
                          activo=false;
                          return;
                          }

espera=(long)((java.lang.Math.random()*5000)+(conteopersona*1000));
                      try {
                          this.sleep(espera);
                      }catch (InterruptedException e) {}
                      Entry llamar=new Entry();
                      llamar=llamada;
                      Entry entrar=new Entry();
                      entrar=entrada;
                      System.out.println("Persona "+conteopersona+"
con espera "+espera+" llama a el ascensor PARA IR DESDE EL PISO
"+origen+" HASTA EL PISO "+destino);
                      System.out.println();
                      Integer
resllam=(Integer)llamar.call(origen,conteopersona);//se manda el piso de
origen y el signo de sentido
                      if (resllam==origen) {
                          System.out.println("
Persona "+conteopersona+" entra en el ascensor en el piso"+origen+" para
ir al piso "+destino);
                          System.out.println();
                          Integer
respdestino=(Integer)entrar.call(destino,conteopersona);
                          if (respdestino==destino) {
                              System.out.println("
Persona "+conteopersona+" que venia del piso "+origen+" sale del
ascensor en el piso "+destino);
                              System.out.println();
                              activo=false;
                          }
                      }
                       if (activo==false){
                          return;
                      }
            }
      }

      public void main(String[] args) {
        }

}
```

CABINA.JAVA

```
package ascensor;

import java.util.*;
import java.lang.*;
```

```
import ascensor.Entry.Mensaje;

public class Cabina extends Thread {
        //private Mensaje m;
      private Entry llamada,entrada;
        public static List Listallamada=new LinkedList();
      public static List Listasubida=new LinkedList();
      public static List Listabajada=new LinkedList();
      public static boolean subida=true;//subir es cierto, bajar es
falso
        public static Integer pisoactual,interior=0;
        public static Integer origen, destino;
        public ListIterator piso; //se crea un iterador
        public ListIterator pisobaja; //se crea un iterador
        public ListIterator pisosube; //se crea un iterador
        static boolean pisoiter,pisosubeiter,pisobajaiter;
        public static TreeMap personas=new TreeMap();
        public static TreeMap peticionpersonas=new TreeMap();
        public static TreeMap destinopersonas=new TreeMap();
        public Cabina(Entry llama,Entry entra) {
            llamada=llama;
            entrada=entra;
      }

        public   void run() {
            pisoiter=false;
            pisosubeiter=false;
            pisobajaiter=false;
            pisoactual=0;
            prepararListas();


            inicializar();

            Entry llamar=new Entry();
            llamar=llamada;
            Entry entrar=new Entry();
            entrar=entrada;

            while (true){
                try {
                    while
((Persona.activeCount()>2)&&(llamada.cola.size()==0)&&(Listallamada.size
()==0)&&(Listabajada.size()==0)&&(Listasubida.size()==0)) {
                        this.sleep(1000);
                    }
                }catch (InterruptedException e) {}
                while (llamada.cola.size()>0) {
                    Mensaje peticiones=(Mensaje)llamar.accept();
                    Integer
origen=java.lang.Math.abs((Integer)peticiones.peticion);
                    Canal
canalrespuesta=(Canal)peticiones.canalRespuesta;
                    Integer person=(Integer)peticiones.persona;
                    if (peticionpersonas.containsKey(origen)) {
                        personas=(TreeMap)peticionpersonas.get(origen);
                    }
                    personas.put(person,canalrespuesta);
                    peticionpersonas.put(origen,personas.clone());
                    personas.clear();
                    if (origen!=null) {
```

```
                        Listallamada.add(origen);//se ponen las llamadas
como enteros positivos
                    }
                }
                java.util.Collections.sort(Listallamada);
                Ascensor();

while((peticionpersonas.containsKey(pisoactual))&&(interior<8)) {
                    TreeMap
personas=(TreeMap)peticionpersonas.get(pisoactual);
                    if (personas.size()==0) {
                        break;
                    }
                        Canal
canalrespuesta=(Canal)personas.get(personas.firstKey());
                        personas.remove(personas.firstKey());

peticionpersonas.put(pisoactual,personas.clone());
                        personas.clear();
                        llamada.reply(pisoactual,canalrespuesta);
                        try {
                            this.sleep(500);
                        }catch (InterruptedException e) {}
                        Mensaje destino=(Mensaje)entrada.accept();
                        Integer
dest=java.lang.Math.abs((Integer)destino.peticion);
                        Canal
canaldestino=(Canal)destino.canalRespuesta;
                        Integer person=(Integer)destino.persona;
                        if (destinopersonas.containsKey(dest)) {
                            personas=(TreeMap)destinopersonas.get(dest);
                        }
                        personas.put(person,canaldestino);
                        destinopersonas.put(dest,personas.clone());
                        personas.clear();
                        interior=interior+1;
                        int actual=piso.nextIndex();
                        Listallamada.remove(pisoactual);
                        if (dest<pisoactual) {
                                Listabajada.add(dest);

java.util.Collections.sort(Listabajada);// se ordenan las llamadas
                            }else if (dest>pisoactual){
                                Listasubida.add(dest);

java.util.Collections.sort(Listasubida);// se ordenan las llamadas
                            }
                 }
                if (destinopersonas.containsKey(pisoactual)) {
                    personas=(TreeMap)destinopersonas.get(pisoactual);
                    while ((personas.size()>0)) {
                        Canal
canaldestino=(Canal)personas.get(personas.firstKey());
                        personas.remove(personas.firstKey());
                        interior=interior-1;
                        entrada.reply(pisoactual,canaldestino);
                        try {
                            this.sleep(500);
                        }catch (InterruptedException e) {}
                        if (Listabajada.contains(pisoactual)) {
                            int actualbaja=pisobaja.nextIndex();
```

```
                            Listabajada.remove(pisoactual);
                        }
                        if (Listasubida.contains(pisoactual)) {
                            int actualsube=pisosube.nextIndex();
                            Listasubida.remove(pisoactual);
                        }
                    }

                }
                try {
                    if (Persona.activeCount()<3) {
                        this.sleep(30000);
                        if (Persona.activeCount()<3) {
                        break;
                        }
                    }
                }catch (InterruptedException e){}
        }
        }

        public   void Ascensor(){
            inicializar();
             if ((Listasubida.size()==0) &&(Listabajada.size()==0)) {
                System.out.println("                    EL ASCENSOR ESTA
VACIO");
                vacio();//el ascensor esta vacio
            }else if
(((Listasubida.size())+(Listabajada.size())==8)==true){
                System.out.println("                    EL ASCENSOR ESTA
LLENO");
                lleno();//el ascensor esta lleno
            }else if
(((Listasubida.size())+(Listabajada.size())<8)==true){
                System.out.println("                    EL ASCENSOR ESTA
MEDIADO");
                mediado();//el ascensor esta mediado
            }
        }


        public   void inicializar() {

            if (Listallamada.contains(pisoactual)) {

piso=Listallamada.listIterator(Listallamada.indexOf(pisoactual));
            }else {

piso=Listallamada.listIterator(ActualizarIterador(Listallamada,pisoactua
l));
            }
            if (Listasubida.contains(pisoactual)) {

pisosube=Listasubida.listIterator(Listasubida.indexOf(pisoactual));
            }else {

pisosube=Listasubida.listIterator(ActualizarIterador(Listasubida,pisoact
ual));
            }
            if (Listabajada.contains(pisoactual)) {

pisobaja=Listabajada.listIterator(Listabajada.indexOf(pisoactual));
```

```
            }else {

pisobaja=Listabajada.listIterator(ActualizarIterador(Listabajada,pisoact
ual));
            }
    }


        public   int ActualizarIterador(List lista,Integer actual) {
            lista.add(actual);
            java.util.Collections.sort(lista);
            int posicion=lista.indexOf(actual);
            lista.remove(actual);
            return posicion;
        }


        public static void IrA(int destino) {


            if ((pisoactual<destino)==true) {
                subida=true;
            System.out.println("                    EL ASCENSOR SUBE
DESDE EL PISO  "+pisoactual+" AL PISO "+destino);
            System.out.println();
            }else if ((pisoactual>destino)==true) {
                subida=false;
            System.out.println("                    EL ASCENSOR BAJA
DESDE EL PISO  "+pisoactual+" AL PISO "+destino);
            System.out.println();

            }else if (pisoactual==destino) {
                System.out.println("                    se queda en piso
final  "+pisoactual);
            }
            pisoactual=destino;

        }


        public static   void prepararListas(){
            Listallamada.clear();
            Listasubida.clear();
            Listabajada.clear();
        }


        public   void vacio() {
            Integer anterior,posterior;
            if (Listallamada.size()>0) {
                if (Listallamada.contains(pisoactual)) {
                    IrA(pisoactual);
                    return;
                }
                if ((piso.hasPrevious()==true)&&(piso.hasNext()==true))
{//existen llamadas de arriba y abajo
                    anterior=(Integer)piso.previous();
                    piso.next();
                    posterior=(Integer)piso.next();
                    piso.previous();
```

```
                    if ((pisoactual-anterior<posterior-
pisoactual)==true) {//esta mas cerca el de abajo
                        IrA(anterior);
                    }else if ((pisoactual-anterior>posterior-
pisoactual)==true) {//esta mas cerca el de arriba
                        IrA(posterior);
                    }else if ((pisoactual-anterior==posterior-
pisoactual)==true) {//los dos iguales, se escoge segun el sentido del
ultimo movimiento
                        if (subida==false) {
                            IrA(anterior);
                        }else {
                            IrA(posterior);
                        }
                    }
                }else if (piso.hasPrevious()==true){ //existen solo
llamadas de abajo
                     IrA((Integer)piso.previous());
                }else if (piso.hasNext()==true){ //existen solo llamadas
de arriba
                     IrA((Integer)piso.next());
                }
            }

        }

        public   void lleno() {
            if (subida==false) {
                if(Listabajada.size()>0) {
                    IrA((Integer)pisobaja.previous());
                }else {
                    IrA((Integer)pisosube.next());
                }
            }else  {
                if(Listasubida.size()>0) {
                    IrA((Integer)pisosube.next());
                }else {
                    IrA((Integer)pisobaja.previous());
                }
            }
        }

        public   void mediado() {
            Integer dentro,fuera=0;
            if (Listallamada.contains(pisoactual)) {
                IrA(pisoactual);
                return;
            }
        boolean b,c,d,e;
        b=piso.hasPrevious();
        c=piso.hasNext();
        d=pisobaja.hasPrevious();
        e=pisosube.hasNext();
        if
(((subida==false)&&(b)&&(d==false))||((subida)&&(b)&&(c==false)&&(d==fal
se)&&(e==false))) {
            subida=false;
            IrA((Integer)piso.previous());
        }else if
(((subida)&&(c)&&(e==false))||((subida==false)&&(b==false)&&(c)&&(d==fal
se)&&(e==false))) {
```

```
            subida=true;
            IrA((Integer)piso.next());
        }else if
(((subida==false)&&(b==false)&&(d))||((subida)&&(b==false)&&(c==false)&&
(d)&&(e==false))) {
            subida=false;
            IrA((Integer)pisobaja.previous());
        }else if
(((subida)&&(c==false)&&(e))||((subida==false)&&(b==false)&&(c==false)&&
(d==false)&&(e))) {
            subida=true;
            IrA((Integer)pisosube.next());
        }else if
(((subida==false)&&(b)&&(d))||((subida)&&(b)&&(c==false)&&(d)&&(e==false
))) {
            subida=false;
            dentro=(Integer)pisobaja.previous();
            fuera=(Integer)piso.previous();
            int comp=dentro.compareTo(fuera);
            if ((comp<0)==true) {
                IrA(fuera);
            }else {
                IrA(dentro);
            }
        }else if
(((subida)&&(c)&&(e))||((subida==false)&&(b==false)&&(c)&&(d==false)&&(e
))) {
            subida=true;
            dentro=(Integer)pisosube.next();
            fuera=(Integer)piso.next();
            int comp=dentro.compareTo(fuera);
            if ((comp>0)==true) {
                IrA(fuera);
            }else {
                IrA(dentro);
            }
        }return;
        }


      public static void main(String[] args) {
      }
}
```

PRINCIPAL.JAVA

```
package ascensor;

import java.lang.*;
import java.lang.Math;
import java.util.*;

public class principal {


    public principal() {
    }

    public static void main(String[] args) {
        Entry llama=new Entry();
        Entry entra=new Entry();
```

```
        for(int i=0;i<=100;i++) {
            int or=(int)java.lang.Math.abs(java.lang.Math.random()*17);
            int des=(int)java.lang.Math.abs(java.lang.Math.random()*17);
            new Persona(llama,entra,or,des).start();
        }
        new Cabina(llama,entra).start();
    }

}
```