ETSI Telecomunicación

Programación Avanzada

Apuntes de Pak

Programación Avanzada

Apuntes de Pak (Fco. J. Rodríguez Fortuño)
ETSI Telecomunicación. Universidad Politécnica de Valencia.
Segundo cuatrimestre de 3er curso
Curso 2005/2006

Contenido

- Referencia muy rápida de conceptos (no hay apuntes extensos)
- Algunos problemas

Fecha de última actualización: 30 Marzo 2008

Tema 2: El lenguaje Java

Constructores:

- Si no ponemos → constructor por defecto
  PERO si ponemos constructor, ya no se puede usar el por defecto
- Si ponemos podemos hacer sobrecarga

```
Caja (double a, double b, double c) {
    ancho = a;
    alto = b;
    prof = c;
}
```

```
Caja (double a) {
    ancho = alto = prof = v;
```

Métodos:

objeto_clase.Metodo(argumentos)

Si son variables: se pasan por valor
Si son objetos o vectores: por referencia
cuidado con lo que hagamos con ellos ⚠

Manejo vectores:

```
int [] vec;   ← o int vec[]
vec = new int [LONG];
for (int i=0; i< vec.length; i++)
    vec[i] = i;
int [] vec2 = {1,2,3,4};
```

se puede con cadenas
`String[] hola = {"hola", "adios", "nos"}`

Vectores de objetos

```
Objeto vec_obj [];
vec_obj = new Objeto[LEN]
vec_obj [i] = new Objeto();
```

Matrices

Cadenas ver pag 54

```
float [][] mat;
mat = new float [LEN1][];
mat [0] = new float [LEN2];
mat [1] = new float [LEN3];
:
```

Lista enlazada

```
LinkedList lista;
lista = new LinkedList();
lista.get(i) ← devuelve object. Hacer casting
lista.add (objeto)
lista.size();
```

Static:

- atributos static
  ↳ se dan valor en bloque static { ... }
- métodos static
  permiten Clase.método(); sin necesidad de objeto creado

- atributos static: son propios de la clase, no de cada objeto de esa clase. si se modifican, se modifican para todas las clases
- atributos final: no pueden modificarse

Sugerencia: como copiar un objeto

- Hacer constructor que acepte como parámetro un objeto de su mismo tipo y vaya copiando todo lo necesario.
- Hacer método estático que devuelva copia del objeto pasado como parámetro

Tema 3- Herencia

```
class Caja {
   int alto, ancho, prof
   métodos
      ⋮
}
```

```
class CajaPeso extends Caja
   int peso
   otros métodos
      ⋮
}
```

- la clase CajaPeso ahora dispone de todos los atributos y métodos de la clase caja.
- El constructor por defecto de CajaPeso ahora llama al de Caja.
- Podemos hacer sobreescritura de métodos: que CajaPeso tenga métodos con mismo nombre que Caja. Quedan sustituidos en el tipo dinámico.
- si hacemos constructor (eliminando así el constructor por defecto) puede ser util llamar al del padre con super()

```
class Caja {
   int alto, ancho, proj;

   Caja (int alto, int ancho, int proj)
      this.alto = alto;
      this.ancho = ancho;
      this.proj = proj;
   }
```

```
class CajaPeso extends Caja {
   int peso;

   CajaPeso (int al, int an, int pr, int peso) {
      super (al, an, pr);
      this.peso = peso;
   }
```

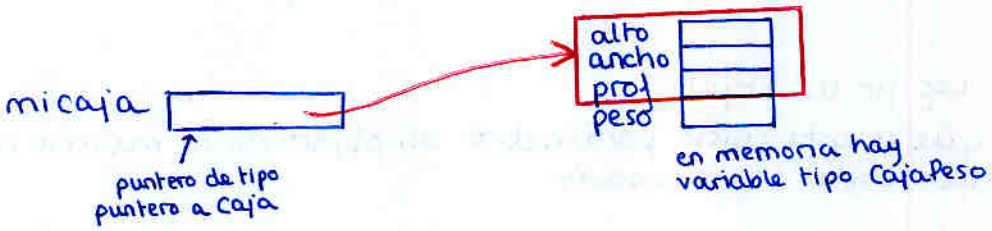
Tipo estático y tipo dinámico

Una variable tipo Caja puede apuntar a cajaPeso

```
Caja micaja = new CajaPeso (1,1,1,3);
```

- Caja: tipo estático (la que declaras)
- CajaPeso: tipo dinámico (a la que apuntas)

⇒ micaja { tipo estático: Caja; tipo dinámico: CajaPeso }



1: Con micaja sólo puedo acceder a atributos y métodos que existan en Caja (i.e. no puedo acceder a Peso)

2: Si con micaja llamo a un método de la clase Caja que la clase CajaPeso sobreescribe, se ejecuta el método sobreescrito en la clase CajaPeso !!!!!

Parece que 1 y 2 se contradigan, pero ahí esta la clave de toda JAVA

Consecuencias:

- casting: se puede hacer sin riesgo a que falle →

```
CajaPeso mipesadacaja = new CajaPeso(...);
Caja micaja;
micaja = (Caja) mipesadacaja
```

tb puede hacerse directamente
micaja = mipesadacaja

- Usar el padre como parámetro a método

```
void QuemarCaja (Caja pobrecaja) {
    pobrecaja.alto = 0;
    pobrecaja.ancho = 0;
    pobrecaja.proj = 0;
}
```

es de tipo CajaPeso, pero el método hará casting
(pobrecaja = (caja) mipesadacaja)

Ahora podemos hacer: Clase.QuemarCaja(mipesadacaja)

- instanceof

```
mipesadacaja instanceof CajaPeso == true
mipesadacaja instanceof CajaColor == false
```

Devuelve true si el tipo del segundo operador coincide con el tipo dinámico (o alguno de los cuales hereda) del primer operador. ↓ (o interfaz q. implementa)

Clases abstractas

pag 64

Una clase de la que queremos que otras hereden pero que nunca queremos que se haga un New de ella

ej: abstract class Figura

```
class Cuadrado extends Figura
class Triangulo extends Figura
:
```

En memoria habrán triángulos, cuadrados, etc... todos referibles mediante tipo estático Figura. Pero no tiene sentido tener un objeto Figura en memoria.

- Si un método es abstract, la clase debe ser abstract
- No se permiten objetos / instancias de clase abstracta (new)
- No se permite abstracto y estático

Un método abstracto es un método que no tiene cuerpo — será implementado por quien herede

```
abstract class Figura {
abstract int calcularArea();
abstract void dibuja(Color c);
}
```

Una clase abstracta puede heredar de otra abstracta, y puede implementar algunos métodos y otros dejarlos como abstractos para quien herede de ella

Nota: Sí se pueden hacer variables de un tipo estático de clase abstracto, para que apunten a tipos dinámicos que hereden. Ello incluye hacer vectores

ej:

```
Figura f[] = new Figura[2]
```

hemos hecho un vector de punteros a Figura, no hemos creado ninguna figura.

la esencia de P.O.O.

```
f[0] = new Triangulo(1,2)
f[1] = new Cuadrado(3)
for(int i=0; i<2; i++)
    f[i].area()
```

Interfaces ver pag 70

Solucionan la ausencia de herencia múltiple
Listan los atributos y métodos que deben tener las clases que implementen la interfaz.

Los variables declaradas son implícitamente static (hay q darles valor)
los métodos no tienen cuerpo

acceso por defecto (publico para este paquete) o publico (no valen otros accesos)

```
interface nombre_interface {
    tipo var1 = valor;   } → hay que darles valor (implicitamente static)
    tipo var2 = valor;   }
    tipo metodo1 (...);   ← no hay llaves ni cuerpo
    tipo metodo2 (...);
}
```

Toda clase que implemente la interfaz (class MiClase implements nombre_interface)
- debe tener todos los métodos de la interfaz a no ser que sea abstracta (en cuyo caso puede dejar el método para las hijas)
- los métodos de la interfaz que implementa deben ser public ⚠

Puedo referenciar a los objetos que implementan la interfaz con una variable cuyo tipo estático es la interfaz

Paquetes y modificadores de acceso ver pag 72

paquete: equivale al directorio donde se guardaran las clases .class al compilar

```
package mi_paquete;
: a partir de aquí todo esta en el paquete
```

- Pueden haber clases con el mismo nombre en distinto paquete.
- se puede anidar paquetes

Control de acceso: (nota: subclase = herencia)

private: SÓLO puede acceder esta misma clase; ni siquiera las subclases

sin modificador (no poner nada): TODAS las clases de SÓLO este paquete

protected: TODAS las clases de este paquete y además subclases de diferente paquete

resulta curioso que protected es más permisivo que no poner nada

public: TODAS las clases de TODOS los paquetes

Tema 4 - Excepciones pag 75

```
try {
    b = 42/a;
    try {
        a = a / (a-2);
        vec [a] = 3;
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println ("Fuera de limites");
    }
} catch (ArithmeticException e) {
    System.out.println ("División por cero");
    throw e;
} catch (Exception e) {
    ...
}
```

divide by zero
divide by zero
out of bounds
lo caza
no lo caza
se propaga
la relanza la caza el siguiente que la pille
Exception caza todas

- Hay instrucciones que pueden lanzar excepciones.
- Una vez lanzadas, la ejecución SALTA todo el bloque try y se va uno a uno a los catch a ver cual es el primero que la caza
  ⇒ El orden de los catch importa ⇒ Nunca poner fuera del try código que dependa de lo de dentro, ya q puede no haberlo hecho!!!
- Si no la caza ningún catch la ejecución da error y se detiene
- Si la caza un catch, hace lo que diga el catch, y sigue la ejecución FUERA del TRY

Propagación de excepciones

- Cuando una excepción no es cazada, se propaga a "la jerarquia superior"
- Si llega al final del método puede propagarse al lugar donde se llamó al método, si el método "throws" ese tipo de excepción

ejemplo

```
class Hola {
  public static void main () {
    try {
      dividir (4,0);
    } catch (Exception e) {
      ...
    }
  }
}
```

aquí al fin se captura

```
class Math {
  void dividir (float a, float b) throws
      ArithmeticException {
    a / b;
  }
```

genera excepción
se lanza a donde se llamó al método
llega al final del método, el cual SABE lanzarla

Excepciones propias (ver práctica de sistema autónomo)

```
class NoNatural extends Exception { }

class Operaciones{
    public int CogerNatural() throws NoNatural {
    try {
        :
    } catch ( Exception e ){
        if (i<=0) throw new NoNatural();
    }
}
```

nosotros mismos comprobamos la condicion de la excepción

Ahora en cualquier parte del programa podemos usar el método CogerNatural() sabiendo que puede lanzar la excepción NoNatural.

Tema 6. Applets

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
(import java.util.LinkedList;) ← típico
```

Código HTML para incluir Applet:

```
<HTML>
  <HEAD>
  </HEAD>
  <BODY>
    <APPLET code="MiApplet.class" WIDTH=400 HEIGHT=400>
    </APPLET>
  </BODY>
</HTML>
```

Componentes de interfaz de usuario (los más típicos)

Applet

```
applet.showStatus("Barra de estado");
```

Label

```
Label label_titulo;
init {
    label_titulo = new Label("Bienvenido");
                   ("Bienvenido", Label.RIGHT);
}
label_titulo.setText("LaLA");
            .getText()
```

Choice

```
choice1 = new Choice();
choice1.addItem("Rojo");
       .remove(int)
       .remove(String)
string = choice1.getSelectedItem();
               .getSelectedIndex();
```

tiene su .addItemListener(escuchador)
escuchador implements ItemListener
public void itemStateChanged(ItemEvent e)

Checkbox

```
Checkbox check = new Checkbox("Label", false)
bool = check.getState();
```

← estado inicial (optativo)

List

```
List lista = new List(5);
```

← elementos visibles sin tener que scrollear

```
lista.add(string)
string = lista.getItem(int)
               .getSelectedItem();
```

.removeAll()

Button

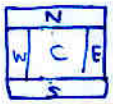
```
Button b = new Button("Inicio");
b.getLabel();
b.setLabel("Parada");
```

Gestores de ubicación

1 · se crea el Layout y se asigna a un Applet, Panel, etc...

2 · Ahora el método add del Applet, Panel, etc... se encargará de todo sin volver a tocar nosotros el Layout

BorderLayout




```
BorderLayout b_layout = new BorderLayout( );
applet. setLayout ( b_layout );
applet. add ( "North", boton_inicio )
applet. add ( "south", label_texto )
```

1

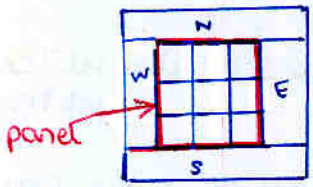
2

Grid Layout

```
Panel panel1 = new Panel ( );
GridLayout grid = new GridLayout (3,3);
panel1. set Layout ( grid );
panel1. add ( boton1 );
panel1. add ( boton2 );
    ⋮
```

se va rellenando la matriz

applet. add ("center", panel1) → Vemos como quedaría así



```
1   import java.awt.*;
2   import java.awt.event.*;
3
4   public class listener1 extends java.applet.Applet {
5       Button b;
6       EscuchaAcciones escuchador;
7       EscuchaRaton escucharaton;
8
9       public void init() {
10          b=new Button("Pulsame");
11          add(b);
12
13          escuchador=new EscuchaAcciones();
14          escucharaton=new EscuchaRaton(this);
15
16          b.addActionListener(escuchador); // Al boton
17          addMouseListener(escucharaton); // Al applet
18      }
19
20
21  }
22
23  class EscuchaAcciones implements ActionListener {
24      public void actionPerformed(ActionEvent e) {
25      ((Button)e.getSource()).setLabel("Muy Bien");
26      }
27  }
28
29
30  class EscuchaRaton implements MouseListener {
31      java.applet.Applet ap;
32
33      EscuchaRaton(java.applet.Applet applet) {
34      ap=applet;
35      }
36
37      // Accion de Hacer Click sobre el applet
38      public void mousePressed(MouseEvent e) {
39          ((java.applet.Applet)e.getSource()).showStatus("x="+e.getX()+"
    y="+e.getY());
40          ap.showStatus("x="+e.getX()+"  y="+e.getY());
41      }
42      // Y ahora todos los otros métodos de la interfaz MouseListener
43      public void mouseReleased( MouseEvent e) {;}
44      public void mouseClicked( MouseEvent e) {;}
45      public void mouseEntered( MouseEvent e) {;}
46      public void mouseExited( MouseEvent e) {;}
47
48  }
49
50
51
52
```

En clase hemos visto 3 formas de hacer los escuchadores.
Aqui están las 3

```
1   // ADEMAS Eventos de Raton sobre el Applet
2
3   import java.awt.*;
4   import java.awt.event.*;
5
6   public class listener2 extends java.applet.Applet implements ActionListene
7
    MouseListener {
8       Button b;
9
10      public void init() {
11          b=new Button("Pulsame");
12          add(b);
13
14          b.addActionListener(this);
15          addMouseListener(this);
16      }
17
18      // Action de pulsar el boton b
19      public void actionPerformed(ActionEvent e) {
20          ((Button)e.getSource()).setLabel("Muy Bien");
21      }
                ↳ getSource devuelve Object
22
23      // Accion de Hacer Click sobre el applet
24      public void mousePressed(MouseEvent e) {
25          showStatus("x="+e.getX()+"  y="+e.getY());
26      }
27      // Y ahora todos los otros métodos de la interfaz MouseListener
28      public void mouseReleased( MouseEvent e) {;}
29      public void mouseClicked( MouseEvent e) {;}
30      public void mouseEntered( MouseEvent e) {;}
31      public void mouseExited( MouseEvent e) {;}
32
33
34  }
35
```

```
29
28
27
26  }
25      }
24          });
23      }
22          showStatus("x="+e.getX()+"  y="+e.getY());
21      public void mousePressed(MouseEvent e) {
20      addMouseListener(new MouseAdapter() {
19
18
17          });
16      }
15          b.setLabel("Muy Bien");
14      public void actionPerformed(ActionEvent e) {
13      b.addActionListener(new ActionListener() {
12
11          add(b);
10      b=new Button("Pulsame");
9   public void init() {
8
7       Button b;
6   public class Listener3 extends java.applet.Applet {
5
4   import java.awt.event.*;
3   import java.awt.*;
2
1   // Clases Anidadas
```

↳ las clases anidadas pueden ver todo lo del applet. Incluso atributos privados

Cuestión 4 (1 pto): Realizad los cambios necesarios para que los métodos insertar y extraer generen las excepciones PilaLLena y PilaVacia respectivamente. Así como añadir al programa principal el código necesario para que se muestre por pantalla lo siguiente:

Insertar 12 elementos

Pila LLena

Pila LLena

Extraer 12 elementos

Pila Vacia

Pila Vacia

```
class PilaLLena extends Exception { }
class PilaVacia extends Exception { }
```

```
class Pila {
   int v[];
   int tam;
   int pos;

   Pila(int t) {
      tam=t;
      v=new int[t];
      pos=-1;
    }

   void insertar(int e) {
     if (pos==tam-1) return;
     v[++pos]=e;
   }
   int extraer() {
     if (pos>=0) return v[pos--];
     return -1;
   }
}
```

```
void insertar(int e) throws PilaLLena {
  if (pos == tam-1)
      throw new PilaLLena();
  v[++pos] = e;

int extraer() throws PilaVacia {
   if (pos >=0)
       return v[pos--];
   throw new PilaVacia();
}
```

```
class excepciones {
    public static void main(String args[]) {
     Pila p=new Pila(10);
     int i;
     int v;

     System.out.println("Insertar 12 elementos");
     for(i=0;i<12;i++){
          p.insertar(i);
     }

     System.out.println("Extraer 12 elementos");
     for(i=0;i<12;i++){
         v=p.extraer();
     }
    }
}
```

```
try {
  p.insertar(i);
} catch (PilaLLena e) {
  ...println("Pila Llena");
}
```

```
try {
   v = p.extraer();
} catch (PilaVacia e) {
   ...println("Pila Vacia");
}
```

Mayor dificultad,
que se muestre por pantalla

Insertar 12 elementos
PilaLlena 1
PilaLlena 2
Extraer 12 elementos
Pila Vacia 3
Pila Vacia 4
↑
cuenta de excepciones que ha llevado

utilizando en el println

println (e)

¡Comparten la cuenta! Necesitaré clase padre común con atributo cont.

```
class PilaOutOfBounds extends Exception {
    static int cont;
    static { cont = 0; }
    PilaOutOfBounds() {
        cont++;
    }
```

↑ es el constructor por defecto

```
class PilaLlena extends PilaOutOfBounds {

    public String toString() {
        String s;
        s = "Pila Llena" + cont;
        return s;
    }
```

← no necesita constructor, usará el de por defecto.

El constructor por defecto hace super del constructor por defecto del padre.

Examen de Programación Avanzada: Cuestiones

DSIC—ETSIT

13 de Junio de 2005

ATENCIÓN: Cada cuestión en hojas separadas.

Cuestión 1 (1.5 pto): Se quiere implementar una clase que almacene una serie de procesos (conjunto de instrucciones que deben ejecutarse en la CPU de un ordenador).

Cada proceso podrá ser de dos tipos: de Entrada/Salida y de Cómputo. Cada proceso contiene:

- Un atributo `estado` de tipo entero que indica el estado en que se encuentra actualmente el proceso (1=PARADO, 2=ESPERA, 3=EJECUCION).
- Un método `ejecutar` que muestra por pantalla uno de los siguientes mensajes:
  - `Proceso de Entrada/Salida en ejecución`
  - `Proceso de Cómputo en ejecución`

  según se trate de un proceso de E/S o de Cómputo. Además, pone el `estado` a EJECUCION (sin importar el estado en que estuviera anteriormente).

La clase `ConjuntoProcesos` heredará de `LinkedList` (clase YA existente en Java). Esta clase contendrá un método `ejecutar` que deberá invocar a su vez el método `ejecutar` de cada uno de sus procesos. Además contiene (entre otros) los siguientes métodos heredados de `LinkedList`:

```
void add( Object obj ); // Añade un objeto a la lista de procesos
Objet get( int i );     // Obtiene el i-ésimo proceso del conjunto (en forma
                           de Object),
int size();             // Devuelve el número de procesos del conjunto.
```

Se pide implementar el código necesario para que el siguiente programa funcione correctamente:

```
class Cuestion {
   public static void main(String args[]) {
      ConjuntoProcesos cp = new ConjuntoProcesos();
      cp.add( new EntradaSalida( Proceso.ESPERA ) );
      cp.add( new Computo( Proceso.ESPERA ) );
      cp.add( new EntradaSalida( Proceso.PARADO ) );
      cp.add( new Computo( Proceso.PARADO ) );

      // En general se debe poder añadir un número indeterminado de procesos
      // de cualquiera de los dos tipos permitidos y en cualquiera de los
      // tres estados permitidos.

      cp.ejecutar();
   }
}
```


P1

```
abstract Class Proceso {
        static final int ESPERA = 2;
        static final int PARADO = 1;
        static final int EJECUCION = 3;

        int estado;
        abstract ejecutar();

        Proceso ( int estado) {
            this.estado = estado;
        }
}
```

```
static int PARADO
static int ESPERA
static int EJECUCION

static {
    PARADO = 1;
    ESPERA = 2;
    EJECUCION = 3;
}
```

```
Class EntradaSalida extends Proceso {
        EntradaSalida (int estado) {
            super (estado);
        }
        void ejecutar () {
            this.estado = EJECUCION;
            System.out.println ("Proceso de E/S en ejecucion");
        }
}

Class Computo extends Proceso {
        Computo (int estado) {
            super (estado);
        }
        void ejecutar () {
            this.estado = EJECUCION;
            System.out.println ("Proceso de Computo en ejecucion");
        }
}

Class ConjuntoProcesos {

    void ejecutar () {

        for (int i=0; i < this.size(); i++) {
            ((Proceso) get(i)).ejecutar();
        }
```

```
object o;
for (int i=0; i<size(); i++){
    o = get(i);
    ((Proceso)o).ejecutar();
}
```

Cuestión 3 (1 pto): Dada la siguiente implementación de la clase Pila de enteros, se requiere hacer dos método estáticos.

El primero de ellos debe de realizar la unión de dos pilas. La unión de las pilas p1 y p2 da como resultado una pila, p3, con un tamaño total que es la suma de los tamaños de ambas pilas. Los elementos en p3 aparecen en el siguiente orden: primero los elementos de p1 y luego los de p2.

El segundo método estático debe de realizar la diferencia de dos pilas. La diferencia de la pila p1 menos la pila p2 es una nueva pila en la que se encuentran todos los elementos de p1 excepto aquellos que están en p2. El orden de los mismos es el que tenían en la pila p1.

```
class Pila {
    int v[];
    int tam;
    int pos;

    Pila(int t) {
        tam=t;
        v=new int[t];
        pos=-1;
    }

    void insertar(int e) {
        if (pos==tam-1) return;
        v[++pos]=e;
    }
    int extraer() {
        if (pos>=0) return v[pos--];
        return -1;
    }
}
```

```
static Pila Unir (Pila p1, Pila p2) {
    Pila p3;
    p3 = new Pila(p1.tam + p2.tam);
    for (int i=0; i <= p1.pos; i++) {
        p3.insertar(p1.v[i]);
    }
    for (int i=0; i <= p2.pos; i++) {
        p3.insertar(p2.v[i]);
    }
    return p3;
}
```


P3

```
boolean esta (int e) {
      boolean found = false;
      for (int i=0; i<= pos; i++) {
          if (v [i]== e) found = true;
          return enc;
      }
}

static Pila Diferencia (Pila p1, Pila p2) {
    Pila  p3 = new Pila (p1.tam);
    for (int i=0; i<= p1.pos ; i++) {
        if (p2.esta (p1.v[i]) == false)
            n.insertar ( p1.v[i]);
    }
    return n;
}
```

Nota: la intersección sería igual pero con true aquí.

Problema
========

Se pide crear las clases necesarias para implementar el siguiente problema:

Se desea informatizar una plantilla de trabajadores. Los trabajadores pueden ser autónomos o de plantilla. Sean como sean todos tienen en común los siguientes atributos con sus respectivos métodos para consultarlos:

* Nombre
* DNI

Además los autónomos tienen:

* Código de autónomo
* Horas realizadas

Y los de plantilla tienen:

* Fecha de incorporación
* Puesto ocupado

Para dar de alta un trabajador autónomo o de plantilla se deben de proveer todos los datos necesarios.

Así mismo se pide realizar un programa principal que cree instancias de dos trabajadores:

- El primero de ellos un trabajador autónomo llamado "Pedro", con 3.5 horas realizadas, código "B-45", dni "12341234-H".

- El segundo un trabajador de plantilla llamado "Manuel", con dni "34345656-P", fecha de incorporación "12-2-2003" y Puesto ocupado "Director".

Y a continuación consulte los diferentes atributos de los trabajadores creados.

Cuestiones
==========

1. ¿ Tendría sentido crear un objeto de tipo trabajador ?

2. ¿ El siguiente código en el main sería correcto ?

```
Trabajador t[]=new Trabajador[100];

for(int i=0;i<25;i++) t[i]=new Autonomo(...);

for(int i=25;i<50;i++) t[i]=new Plantilla(...);
```

3. ¿ Que ocurre en estas dos instrucciones ?

```
t[0].Nombre();
t[30].Nombre();
```

```
Class Trabajador {
        String nombre;
        String dni;
        Trabajador (String nom, string dn) {
              nombre = nom;
              dni = dn;
        }
        String Nombre (){
              return nombre;
        }
        String DNI (){
              return dni;
        }
}

class Autonomo extends Trabajador {
        String codigo;
        float horas;
        Autonomo (string nom, String dn, String cod, float hor){
              super (nom, dn);
              codigo = cod;
              horas = hor;
        }
        String Codigo(){ return codigo; }
        float Horas (){ return horas; }
}
class Plantilla extends Trabajador {
        String fecha;
        String puesto;
        Plantilla (String nom, string dn, String fech, String puest){
              super (nom, dn);
              fecha = fech;
              puesto = puest;
        }
        String Fecha(){ return fecha;}
        String Puesto (){ return puesto; }
}

class Problema Plantilla {
        public void static main (String args[]){
              Autonomo aut1 = new Autonomo ("Pedro", "12341234-H",
                                            "B-45", 3.5);
              Plantilla plant1 = new Plantilla ("Manuel","34345656-P",
                                            "12-2-2003", "Director");
              System.out.println (aut1.Nombre());
              System.out.println (aut1.Codigo());
        }
}
```

Examen de Programación Avanzada: Problemas

DSIC—ETSIT

13 de Junio de 2005

ATENCIÓN: Los dos problemas se entragán por separado.

Problema 1 (2.5 ptos)

Applet con Lista de Redes y Lista de Hosts asociada

Se dispone de las siguientes clases en Java **YA** implementadas:

- `class Direccion`

  Descripción: Almacena una direccion IP del tipo XXX.YYY donde XXX es un entero que representa la dirección de la red e YYY es otro entero que representa la dirección del host dentro de la red.

  Métodos:

```
Direccion ( int direccionRed, int direccionHost ); // Constructor
String toString(); // Devuelve la direccion IP en forma de String
```

- `class Host`

  Descripción: Representa un host de una red dada. Contiene un atributo de tipo `Direccion`

  Métodos:

```
Host (Direccion dir); //Constructor
Direccion getIP();    // Devuelve la direccion del Host
```

- `class Red extends LinkedList`

  Descripción: Conjunto de Hosts. Contiene un atributo de tipo `Direccion` (direccion genérica de la red). Dicha dirección es de la forma XXX.0

  Métodos:

```
void añadirHost (Direccion dir) throws NoSePuedeAñadirException;
// Crea un objeto de tipo host a partir de la dirección dada y lo añade
// a la red. Lanza la excepción NoSePuedeAñadirException si se produce
// cualquier tipo de error (dirección incorrecta, demasiados hosts...)

Direccion getIP();       // Devuelve la direccion de la Red

Métodos heredados de LinkedList
```

- `class SistemaAutonomo extends LinkedList`

  Descripcción: Conjunto de redes

  Métodos:


```
SistemaAutonomo(); // Constructor. En el propio constructor se crean
                   todas las redes que lo componen, con sus
                   respectivos hosts.
Métodos heredados de LinkedList
```

- `class LinkedList`

  Descripción: Almacena un conjunto de objetos.

  Métodos:

```
void add( Object obj );  // Añade un objeto al conjunto
Object get( int i );     // Devuelve el objeto i-ésimo del conjunto
int size();              // Devuelve el número de objetos del conjunto
```

Se pide:

Crear un applet que cree un objeto de tipo `SistemaAutonomo` y que contenga los siguientes componentes:

- Una lista desplegable (`Choice`) con las direcciones IP de cada una de las redes del Sistema Autónomo.
- Una lista (`List`) con las direcciones IP de los hosts pertenecientes a la red actualmente seleccionada en el `Choice`.
- Dos cuadros de texto (`TextField`) que servirán para introducir una dirección IP (identificador de red e identificador de host dentro de la red respectivamente).
- Un botón (`Button`) que sirva para añadir un host. Cuando se pulse, se leerá la información de los `TextField`, y se añadirá un host a la red actualmente seleccionada. Deberá actualizarse el `List` con el nuevo Host. Tened en cuenta las posibles excepciones que puedan producirse.

El interface deberá ajustarse al mostrado en la siguiente figura:

p5

Se supondrá que el Sistema Autónomo contiene al menos una red, y que cada red contiene al menos un host. Todos los elementos del Sistema Autónomo se crean automáticamente en el constructor del mismo.

SOLUCIÓN:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.LinkedList;

public class GUI_SistemaAutonomo extends Applet implements ActionListener,
                                                            ItemListener {

    Choice    chRed;
    List      lHosts;
    Button    bAnyadir;
    TextField tfIPRed, tfIPHost;
    SistemaAutonomo sa;

    public void init() {
```



```
        // Inicializar Sistema Autonomo
        sa = new SistemaAutonomo();

        // Crear controles
        chRed = new Choice();
        lHosts = new List();
        bAnyadir = new Button("Anadir Host");
        tfIPRed = new TextField(3);
        tfIPHost = new TextField(3);

        // Rellenar Choice
        for( int i = 0; i < sa.size(); i++ ) {
            Object elemento = sa.get(i);
                chRed.add( ((Red)elemento).getIP().toString() );
        }

        // Rellenar Lista
        Red red = (Red)sa.get(0);
        for( int i = 0; i < red.size(); i++ ) {
            Host h = (Host)red.get(i);
            lHosts.add( h.getIP().toString());
        }

        // Establecer Layout
        setLayout( new BorderLayout() );

        // Crear panel central
        Panel pCentral = new Panel();
        pCentral.add(chRed);
        pCentral.add(lHosts);

        // Crear panel inferior
        Panel pInferior = new Panel();
        pInferior.add(new Label("Dir. Red"));
        pInferior.add(tfIPRed);
        pInferior.add(new Label("Dir. Host"));
        pInferior.add(tfIPHost);
        pInferior.add(bAnyadir);

        // Añadir paneles al applet
        add("Center", pCentral);
        add("South", pInferior);

        // Conectar escuchadores
        chRed.addItemListener(this);
        bAnyadir.addActionListener(this);
    }

    public void itemStateChanged( ItemEvent e ) {
        if ( e.getSource() == chRed ) {
```

```
            lHosts.removeAll();
            Red red = (Red)sa.get(chRed.getSelectedIndex());
            for( int i = 0; i < red.size(); i++ )
                lHosts.add((((Host)red.get(i)).getIP().toString()));
        }
    }

    public void actionPerformed( ActionEvent e ) {
        int indiceRed = chRed.getSelectedIndex();
        int dirRed;
        int dirHost;

        try {
            dirRed = Integer.parseInt(tfIPRed.getText());
            dirHost = Integer.parseInt(tfIPHost.getText());

            Direccion d = new Direccion(dirRed, dirHost);
            Red r = ((Red)sa.get(indiceRed));
            r.anyadirHost( d );
            lHosts.add(d.toString());
        } catch ( Exception ex ) {
            showStatus("Error añadiendo host");
        }
    }
}
```


Problema Empresa de Alquiler

Problema 2 (2.5 ptos)

Vamos a intentar simular la gestión de una empresa que alquila vehículos (algunos de ellos con conductor) de toda clase como enseguida veremos. El objetivo del problema es que se definan las diferentes clases e interfaces y las relaciones entre ellas; es por esto que en la defición del problema aparezcan abstracciones (cuestiones muy abiertas de cara a una implementación definitiva) que sirven para explicitar mejor las relaciones entre ellas.

Por tanto, en la empresa tenemos vehículos de los que, en general, nos interesa saber el año de adquisición y el año en el que deben pasar la siguiente ITV. Para saber la ubicación física del vehículo en todo momento, los vehículos tienen instalado un buscador. Éste va emitiendo una señal que nos sirve para ubicar el vehículo. Lo que interesa saber de este buscador, que todos los vehículos tienen, es el lugar en que se encuentra instalado. Por simplificar asumimos que ese lugar está codificado mediante un número entero.

Lo normal es que cuando hablamos de vehículos pensemos en coches, motos, etc, pero como es una empresa arraigada en una zona agrícola puede alquilar carros de tracción animal e incluso bicis, en cierta forma también de tracción animal. Obviamente a este tipo de vehículos no se les pasa la ITV.

De entre los vehículos motorizados tenemos coches, motos y camiones. En cada uno de ellos, nos interesa guardar información diferente propia de cada uno de ellos. Como se trata de una empresa de alquiler, de todos ellos tenemos que poder tramitar un alquiler y estos trámites son diferentes de los vehículos a motor de los vehículos sin motor, a parte, para cada tipo de vehículo concreto, el trámite es un poco diferente.

La empresa tiene un registro de las personas que trabajan en ella y, en general, nos interesa tener un conjunto de información común del personal con cuestiones concretas del personal administrativo, mecánico y conductor. Es un requisito de la empresa que todos sus empleados tenga el carnet de cualquiera de los vehículos que la empresa posee, sobre todo para poder moverlos en caso de puntas de trabajo. La empresa tiene tres tipos de contratados: conductores, mecánicos y personal administrativo.

Los vehículos pueden alquilarse con conductor (que es personal de la empresa). Está claro que tenemos que tener constancia de quien se ha marchado con qué vehículo y de quién ha hecho el trámite de alquiler.

Ahora bien, el alquiler de vehículos con conductor sólo afecta a coches y camiones, además, sólo los conductores y mecánicos pueden ir como conductores en el alquiler de uno de estos vehículos. Por otro lado, el trámite de alquiler sólo lo pueden hacer miembros del personal administrativo.

Una cuestión fundamental en la empresa dentro de la gestión que estamos planteando es la gestión de los stocks, es decir, de los vehículos que en general tenemos en la empresa disponibles a ser alquilados.

Todos estos vehículos que tiene la empresa se han de guardar en tres lugares diferentes: una nave para los carros, un garage para coches y camiones y el almacen para bicis y motos. De cada uno de estos repositorios nos interesa saber su capacidad máxima y quien (del personal administrativo) está a cargo del lugar para la gestión de los alquileres y qué mecanico para la cuestión de las posibles reparaciones.

Se trata, por tanto de que se definan las diferentes clases e interfaces que deberían componer un sistema que siva para la gestión que acabamos de definir.

SOLUCIÓN:

```
interface g_garage {}
interface g_almacen {}
interface p_conducir {
void conducir ();
}
```

```
class buscador {
int posicion;
int emite () {
return 1;
}
}

abstract class vehiculo {
int a_adqui;
administrativo a;
buscador bcdr;

abstract void Alquila ();
}

abstract class motorizado extends vehiculo{
p_conducir pc;
int itv;
//
}


abstract class nomotorizado extends vehiculo{
//
}

class coche extends motorizado implements g_garage {
// atributos propios de coche

coche (int a_ad, int it, buscador bc) {
a_adqui = a_ad;
itv = it;
bcdr = bc;
}

void Alquila () {;}
}

class camion extends motorizado implements g_garage {
// atributos propios de camión
void Alquila () {;}
}

class moto extends motorizado implements g_almacen {
// atributos de moto
void Alquila () {;}
}

class bici extends nomotorizado implements g_almacen {
// atributos de bici
```



```
void Alquila () {;}
}

class carro extends nomotorizado {
// atributos de carro
void Alquila () {;}
}

abstract class personal {
// atributos comunes
abstract void conducir ();
}


class conductor extends personal implements p_conducir{
// atributos conductor
public void conducir () {;}
}

class mecanico extends personal implements p_conducir{
// atributos conductor
public void conducir () {;}
}

class administrativo extends personal {
// atributos conductor
void conducir () {;}
}


class empresa {
personal empleados[];
nave n;
garage g;
almacen a;
alquilados al;

empresa (int i) {
empleados = new personal[i];
empleados[0] = new conductor();
empleados[1] = new mecanico();
empleados[2] = new administrativo();

n = new nave(10);
g = new garage(10);
a = new almacen(10);
al = new alquilados(10);
}

coche newCoche (int a_ad, int it, buscador bc) {
```

```
return (new coche (a_ad, it, bc));
}

}

class nave {
carro aparcados[];
administrativo a;
mecanico c;

nave (int i){
aparcados = new carro[10];
}
}

class garage {
administrativo a;
mecanico c;
g_garage aparcados[];

garage (int i){
aparcados = new g_garage[10];
}
//
}

class almacen {
g_almacen aparcados [];
administrativo a;
mecanico c;

almacen (int i){
aparcados = new g_almacen[10];
}
}

class alquilados {
vehiculo alquilados[];

alquilados (int i){
alquilados = new vehiculo[10];
}
//
}


public class Problema2005 {
public static void main (String a[]) {
empresa e;
buscador b;
```



```
e = new empresa(3);
b = new buscador();
e.g.aparcados[0] = e.newCoche(1998, 2006, b);


}
}
```


P7

Examen de Programación Avanzada (Problemas)

Convocatorio ordinaria. 10 de Junio de 2002.

Duración: 2 horas.

Problema Applet MóvilSMS

Problema 1.

Se quiere diseñar una pequeña aplicación en Java para enviar mensajes de texto a móviles. La interfaz gráfica del programa debe contemplar los siguientes aspectos: introducción del número de teléfono y del mensaje de texto, cancelación de operación de envío, envío directo del mensaje cuando el usuario entienda que esta en una franja horaria adecuada, almacenamiento temporal de los datos de la comunicación caso de encontrarse en una franja horaria inadecuada, envío de los mensajes almacenados. El aspecto externo de la interfaz debe obedecer al siguiente formato:

Se asume que existe una clase predefinida (que podéis usar) denominada `Conexion`, con las siguientes características:

- Constructor, crea una conexión con un número: `(String numero)`
- Método para enviar un mensaje: `void enviar (String mj)`
- Método para cerrar la conexión: `void cerrar ()`

Para hacer uso de la clase `Conexion` no es necesario importar ninguna librería, se supone que es accesible desde la aplicación.

Se pide:

1.- Diseñar las clases necesarias para implementar la interfaz gráfica de usuario (componentes y oyentes), teniendo en cuenta que la aplicación se va a ejecutar en un navegador. **(1 punto).**

2.- Diseñar las clases necesarias para la modelización del concepto de mensaje de texto **(0.8 puntos).**

3.- Diseño de la página HTML donde se establece la invocación a la interfaz gráfica. **(0.2 puntos)**

Solución.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class IGU extends Applet {

        Label l1= new Label(" N_MÓVIL");
        Label l2= new Label(" MENSAJE TEXTO");
        Label l3= new Label(" RESULTADO");
        Button b1= new Button("ANULAR");
        Button b2= new Button("ENVIAR");
        Button b3= new Button("ALMACENAR");
        Button b4= new Button("ENVIAR LISTA");
        TextField t1= new TextField(10);
        TextArea t2= new TextArea(5,10);
          Pila l_mensajes;

        public void init () {
                add(l1);add(t1);
                add(l2);add(t2);
                add(b1); add(b2);
                add(b3); add(b4);
                    l_mensajes=new Pila();
                OYENTE Oy= new OYENTE();
                b1.addActionListener(Oy);
                b2.addActionListener(Oy);
                    b3.addActionListener(Oy);
                    b4.addActionListener(Oy);
                }

        class  OYENTE implements ActionListener {
                  public void actionPerformed(ActionEvent e) {
                  if (e.getSource()== b1) {
                      // caso de operacion anular
                       t1.setText("");
                       t2.setText("");}
                      else if (e.getSource()== b2) {
                         // se establece la conexion, se envía y se cierra
                          Conexion c= new Conexion(t1.getText());
                                  c.enviar(t2.getText());
                                  c.cerrar();}
                             else if (e.getSource()== b3) {
                          // Pilaar el mensaje en la lista de mensajes
                                  // primero creamos el mensaje
                                          Mensaje   m=   new   Mensaje(t1.getText(),
t2.getText());
                                      l_mensajes.insertar(m);}
                                 else if (e.getSource()== b4) {
                                // enviar todos los mensajes de la lista
                                              Mensaje mj;
                                              int n=0;
                                              while (!l_mensajes.vacia()){

mj=(Mensaje)l_mensajes.borrar_prim();
                                                  Conexion        c=        new
Conexion(mj.n_movil);
                                                  n=n+1;
                                                  showStatus("    enviando    al
movil "+ mj.n_movil);
                                                  c.enviar(mj.texto);
                                              c.cerrar();
                                                  showStatus("  el  numero  de
mensajes son"+n);}}}}

}

class Pila {

      class Nodo {
        Object info;
        Nodo prox;
        /* Constructor de la clase */
        Nodo(Object obj, Nodo ptr){
                info=obj;
                prox=ptr; }
        }

      /*Atributos de la clase Lista*/
      Nodo tope;
      int num_elem;
```

```
/*Constructor de la lista*/
Pila (){
  tope=null;
  num_elem=0; }

/*Insertar un elemento, en la cabeza del Pila*/
void insertar(Object elem ){
  /* debemos crear un objeto de tipo nodo */
  Nodo nuevo=new Nodo(elem, null);
  nuevo.prox=tope;
  tope=nuevo;
  num_elem=num_elem+1;
  }
/* elimina y devuelve el primer elemento en la lista*/
Object borrar_prim (){
  /*si el volumen esta vacio debe de devolver null*/
  if(num_elem==0) return null;
  else {
          Object ob;
          ob= tope.info;
          tope=tope.prox;
          return ob;}}

/* determina si el Pila esta vacio*/
boolean vacia (){
  return (num_elem==0); }

  }


class Mensaje {
/* Atributos */
       String n_movil, texto;
       Mensaje( String n, String t){
              n_movil=n;
              texto=t;}}

// esta clase no esta implementada solo se ha definido lo basico para que
funcione la interfaz
class Conexion {
       Conexion (String n){;}
       void enviar(String t){;}
       void cerrar(){;}}
```

Problema PC dispositivos implementan Buses y Manejadores → son interfaces, no clases. No hay objetos tipo bus.

Problema 2.

Se quiere diseñar una aplicación en Java para simular el comportamiento de un ordenador personal. A continuación se presentan las características de los diferentes componentes de un ordenador.

En primer lugar todo ordenador tiene una CPU. Se caracteriza por la frecuencia de su reloj (en Mhz) y los tamaños de la caché de primer y segundo nivel (en Kilobits). Toda CPU es capaz de emitir señal de reloj y de entubar instrucciones. Existen dos tipos de CPU: LETNI y DMA. En la LETNI se puede ampliar la caché de nivel 1 y en la DMA incrementar la frecuencia del reloj (en Mhz). El proceso de emisión de señales es idéntico para los dos tipos de CPU, siendo diferente para cada tipo el proceso de entubar instrucciones.

La memoria RAM, se caracteriza por su capacidad (en Megas), su tiempo de acceso y las operaciones de lectura y escritura.

Estos ordenadores utilizan una serie de dispositivos para comunicarse con el exterior y para almacenar información. Estos dispositivos internamente se conectan con el resto de componentes a través de manejadores y buses.

Existen dos tipos de manejadores: IDE y SCSI. En el primero de ellos las operaciones básicas son: buscar dispositivos esclavos y dispositivos maestros conectados al manejador, y chequear el estado del mismo. Respecto a los SCSI las operaciones básicas son: emitir y recibir instrucciones. En ambos casos se debe tener en cuenta que cada una de las operaciones es *implementada* de manera diferente para cada dispositivo que se conecta al manejador.

Todos los dispositivos se conectan a un bus. Las operaciones sobre los buses son *implementadas* de manera diferente por cada uno de los dispositivos que se conectan a él. Las operaciones básicas que debe *implementar* cualquier dispositivo conectado al bus son: emisión de datos, emisión de direcciones y emisión de señales de control. Se pueden instalar diferentes tipos de buses, todos ellos caracterizados por la anchura del mismo (en bits) y la velocidad de transmisión (en Mhz). Los tres tipos de buses son: PCI (con anchura de 16 bits y velocidad de 200 Mhz), AGP (con anchura de 32 bits y velocidad de 175 Mhz) y USB (con anchura de 16 bits y velocidad de 8 Mhz). Se diferencian en que cada uno de ellos realiza una acción diferente: sincronizar el bus, resetear el bus y encadenarse al bus respectivamente.

Una vez detallados los mecanismos de unión de los diferentes dispositivos, a continuación se presentan algunos ejemplos de dispositivos que se deben contemplar.

Los discos duros realizan operaciones de lectura, escritura y arranque del giro del disco. El mecanismo de arranque del giro es el mismo para todos los tipos de discos. Por el contrario las operaciones de lectura y escritura dependerán del tipo de disco. Se dispone de dos clases de discos: disco IDE que debe realizar las operaciones básicas del manejador IDE y del bus conectado a él, y disco SCSI que debe realizar las operaciones del manejador SCSI y del bus conectado a él.

Se dispone también de tarjetas de diferentes clases que se pueden conectar a los diferentes buses. Una tarjeta se caracteriza porque tiene una RAM asociada (en Kbits) y que siempre realiza una emisión y/o recepción de información. Por ejemplo, a través de una tarjeta de sonido se emite y recibe sonido, a través de una tarjeta de red se emiten y reciben señales codificadas, etc. Ahora bien, cualquier tipo de tarjeta realiza la r un LED que indica si está funcionando. Para todos los tipos de tarjetas el encendido del LED es similar. Hay cuatro tipos de tarjetas: tarjeta de red (Eth0), tarjeta de sonido, tarjeta de vídeo y tarjeta digitalizadora de imágenes. Las dos primeras se conectan a buses PCI y las dos últimas a buses AGP y USB respectivamente.

Se pide:

Determinar las diferentes clases, clases abstractas e interfaces necesarias para la modelización de las componentes mencionadas, estableciendo la estructura jerárquica que las relaciona, sus atributos y métodos, de estos últimos indicar únicamente el nombre y si son o no abstractos. **.(2 puntos)**

Solución.

```
interface IDE {
      void esclavos();
      void maestro();
```

```
        void testea();
}

interface SCSI {
        void emision();
        void recepcion();
}

interface Bus {
        void datos();
        void direccion();
        void control();
}

interface PCI extends Bus {
        int anchura = 16;
        int velocidad = 8;
        void sincroniza();
}

interface AGP extends Bus {
        int anchura = 32;
        int velocidad = 175;
        void reset();
}

interface USB extends Bus {
        int anchura = 16;
        int velocidad = 200;
        void encadena();
}

abstract class tarjeta {
    int ram;
    abstract void emision_externa();
    abstract void recepcion_externa();
    void emitir_luz() {;}
}

class sonido extends tarjeta implements PCI {
    void emision_externa () {
        System.out.println("Emito sonidos por conectores...");
    }
    void recepcion_externa () {
        System.out.println("Recibo sonidos por micro...");
    }
    public void sincroniza () {
        System.out.println("Me sincronizo");
    }
    public void datos () {
        System.out.println("Trasmito datos al bus como tarjeta");
    }
    public void direccion () {
        System.out.println("Transmito direccion al bus");
    }
    public void control () {
        System.out.println("Transmito señales de control");
    }
    sonido (int r) {
        ram = r;
    }
}

class video extends tarjeta implements AGP {
    void emision_externa () {
        System.out.println("Emito imágenes por conectores...");
    }
    void recepcion_externa () {
        System.out.println("Recibo imágenes por camara...");
    }
    public void reset () {
        System.out.println("Me inicio");
    }
    public void datos () {
        System.out.println("Trasmito datos al bus como video");
    }
    public void direccion () {
        System.out.println("Transmito direccion al bus como video");
    }
    public void control () {
        System.out.println("Transmito señales de control como video");
    }
    video (int r) {
        ram = r;
    }
}

class digitalizadora extends tarjeta implements PCI {
    void emision_externa () {
        System.out.println("Señales para iniciar la digitalizaicón");
    }
    void recepcion_externa () {
        System.out.println("digitos del origen");
    }
    public void sincroniza () {
        System.out.println("Me sincronizo con el bus");
    }
    public void datos () {
        System.out.println("Trasmito datos al bus como tarjeta");
    }
    public void direccion () {
        System.out.println("Transmito direccion al bus");
    }
    public void control () {
        System.out.println("Transmito señales de control");
    }
    digitalizadora (int r) {
        ram = r;
    }
}

class eth10 extends tarjeta implements PCI {
    void emision_externa () {
        System.out.println("Señales de transmisión al cable");
    }
    void recepcion_externa () {
        System.out.println("señales de recepción del cable");
    }
    public void sincroniza () {
        System.out.println("Me sincronizo con el bus");
    }
    public void datos () {
        System.out.println("Trasmito datos al bus como tarjeta");
    }
    public void direccion () {
        System.out.println("Transmito direccion al bus");
    }
    public void control () {
        System.out.println("Transmito señales de control");
    }
    eth10 (int r) {
        ram = r;
    }
}
```

```
abstract class HD {
    int va, capacidad;
    abstract void lectura();
    abstract void escritura();
    void giro(){;}
}

class DiscoIDE extends HD implements IDE, Bus {
    void lectura() {
        System.out.println("Leo disco IDE");
    }
    void escritura() {
        System.out.println("Escribo disco IDE");
    }
    public void maestro() {
        System.out.println("Busco maestro en el bus");
    }
    public void esclavos() {
        System.out.println("Busco esclavo en el bus");
    }
    public void testea() {
        System.out.println("Testeo el bus");
    }
       public void datos () {
        System.out.println("Trasmito datos al bus como tarjeta");
    }
    public void direccion () {
        System.out.println("Transmito direccion al bus");
    }
    public void control () {
        System.out.println("Transmito señales de control");
    }

    DiscoIDE (int v, int c) {
        va = v;
        capacidad = c;
    }
}

class DiscoSCSI extends HD implements SCSI, Bus {
       public void datos () {
        System.out.println("Trasmito datos al bus como tarjeta");
    }
    public void direccion () {
        System.out.println("Transmito direccion al bus");
    }
    public void control () {
        System.out.println("Transmito señales de control");
    }


       void lectura() {
        System.out.println("Leo disco SCSI");
    }
    void escritura() {
        System.out.println("Escribo disco SCSI");
    }
    public void emision() {
        System.out.println("Emito comando al bus");
    }
    public void recepcion() {
        System.out.println("Recibo comando del bus");
    }
    DiscoSCSI (int v, int c) {
        va = v;
        capacidad = c;
    }
}
```

```
abstract class UnidadExterna {
    abstract void arranque();
    abstract void parada();
    void recibe_energia(){;}
}

class CDROM extends UnidadExterna implements IDE, PCI {
      void arranque() {
            System.out.println("Arranco el CD");
      }
      void parada() {
            System.out.println("Paro el CD");
      }
    public void maestro() {
        System.out.println("Busco maestro en el bus");
    }
    public void esclavos() {
        System.out.println("Busco esclavo en el bus");
    }
    public void testea() {
        System.out.println("Testeo el bus");
    }
    public void sincroniza () {
        System.out.println("Me sincronizo");
    }
            public void datos () {
        System.out.println("Trasmito datos al bus como tarjeta");
    }
    public void direccion () {
        System.out.println("Transmito direccion al bus");
    }
    public void control () {
        System.out.println("Transmito señales de control");
    }

}

class Scanner extends UnidadExterna implements SCSI, PCI {
      void arranque() {
            System.out.println("Arranco el escáner");
      }
      void parada() {
            System.out.println("Paro el escáner");
      }
    public void emision() {
        System.out.println("Emito comando al bus");
    }
    public void recepcion() {
        System.out.println("Recibo comando del bus");
    }
    public void sincroniza () {
        System.out.println("Me sincronizo");
    }
            public void datos () {
        System.out.println("Trasmito datos al bus como tarjeta");
    }
    public void direccion () {
        System.out.println("Transmito direccion al bus");
    }
    public void control () {
        System.out.println("Transmito señales de control");
    }

}

class DVD extends UnidadExterna implements SCSI, PCI {
      void arranque() {
```

```
            System.out.println("Arranco el escáner");
        }
        void parada() {
            System.out.println("Paro el escáner");
        }
    public void emision() {
        System.out.println("Emito comando al bus");
    }
    public void recepcion() {
        System.out.println("Recibo comando del bus");
    }
    public void sincroniza () {
        System.out.println("Me sincronizo");
    }
            public void datos () {
        System.out.println("Trasmito datos al bus como tarjeta");
    }
    public void direccion () {
        System.out.println("Transmito direccion al bus");
    }
    public void control () {
        System.out.println("Transmito señales de control");
    }
}

class RAM {
        int megas;
        float velo;
        void leer(){;}
        void escribir(){;}
        RAM (int mgs, int v) {
            megas = mgs; velo = v;
        }
}

abstract class CPU {
        int Mhz;
        int cache1, cache2;
        void emite_reloj(){;}
        abstract void entuba();
}

class LETN1 extends CPU {
        int incr;
        void entuba(){;}
}

class LMA extends CPU {
        int incr;
        void entuba(){;}
}
```

Examen de Programación Avanzada: Problemas

DSIC—ETSIT

14 de Septiembre de 2004

ATENCIÓN: Los dos problemas se entragán por separado.

Problema 1. BuscaMinas (2.25 ptos)

BuscaMinas

El buscaminas es un juego cuyo objetivo es localizar todas las minas que se encuentran ocultas en un tablero de NxM celdas. Inicialmente todas las celdas están ocultas, de modo que no se sabe cuáles de ellas contienen una mina cuáles no. El jugador va escogiendo celdas para inspeccionar. Cada vez que escoge (se pisa) una celda, se pueden dar tres situaciones distintas:

1. La celda "pisada" contiene una mina. En este caso se muestran todas las celdas que contenían una bomba y el jugador pierde la partida.
2. La celda "pisada" no contiene una mina:
   a) Alguna de las celdas vecinas (las 8 adyacentes) contienen una mina. En este caso, en la celda pisada se muestra el número de celdas vecinas que albergaban una mina.
   b) Ninguna de las celdas vecinas contiene una mina. En este caso cada una de estas celdas vecinas que no hubieran sido pisadas (seleccionadas) en alguna jugada previa, se pisan de forma automática. Este proceso se repite recursivamente para cada una de las celdas vecinas que resulten no tener tampoco ninguna mina alrededor.

El ejercicio del buscaminas se ha implemnetado con el siguiente applet:

```
public class BuscaMinas extends Applet {
   Tablero tablero;
   Label label;

   public void init() {
      setLayout(new BorderLayout());
   }

   public void start() {
      if( tablero!=null ) remove(tablero);
      if( label!=null ) remove(label);
      tablero = new Tablero(this);
      add(tablero,"Center");
      label = new Label("",Label.CENTER);
      add(label,"South");
   }
}
```

El problema consiste, a partir del código siguiente, implementar las clases Tablero y Casilla descritas a continuación:

clase Tablero:

Esta clase posee una serie de constantes: el número de filas (10) y de columnas (10) del tablero y el número de bombas (15). Posee un contador de casillas pisadas que cuando vale 85 (en este caso particular) determina que el juego ha terminado con la victoria del jugador.

El tablero posee también una referencia a la clase BuscaMinas, mediante la cual accederá a la etiqueta de dicha clase para indicar que el jugador ha ganado o ha perdido. La última variable que posee la clase Tablero consiste en una matriz cuadrada de 10x10 objetos de tipo Casilla.

A continuación se describen los métodos de esta clase:

Tablero: Constructor de la clase. Tiene como objeto establecer el Layout para el Tablero e inicializarlo con las casillas. Para cada coordenada del Tablero se instancia una Casilla y se coloca en el Tablero. Las casillas se inicializan con bomba o sin ella dependiendo del valor de retorno del método **aleatorio**. Una vez colocadas todas las casillas en el Tablero, hay que recorrer éste para contar el número de bomas que posee cada casilla a su alrededor. A cada Casilla se le asigna este número mediante la llamada al método **etiquetar** de la clase Casilla (descrita más adelante).

boolean aleatorio(int i, int j): Este método decide aleatoriamnete si la casilla (i,j) debe contener una bomba. Este método NO hay que implementarlo.

int bombasAlrededor(int i, int j): Este método de la clase Tablero cuenta y devuelve el número de bombas que hay alrededor de la Casilla (i,j).

void mostrarBombas(): Este método recorre todas las casillas del Tablero y, para cada una de ellas, llama al método mostrarBomba() de la clase Casilla. Además, debe mostrar en la etiqueta de la clase BuscaMinas el mensaje: "Lo siento, has perdido".

void vacio(int i, int j): Este método se encarga de "pisar"las casillas de alrededor de la Casilla (i,j) llamando al método pisado() de la clase Casilla.

void actualizarPisadas(): Este método incrementa en 1 el contador de casillas pisadas y comprueba que dicho número sea menor que el número de casillas que no tienen bomba. En el momento que sea igual al número de casillas sin bomba mostrará en la etiqueta de la clase BuscaMinas el mensaje: "Enhorabuena! has ganado".

clase Casilla:

La clase Casilla es de tipo Panel y puede contener una etiqueta o un botón en la misma ubicación. Inicialmente, las casillas muestran un botón y, si el usuario la pisa, mostrarán la etiqueta. La clase Casilla necesitará diversas variables entre las que figuran una referencia a la clase Tablero para poder llamar a sus métodos, el número de bombas que existe alrededor de dicha Casilla, las coordenadas de dicha Casilla dentro del Tablero, dos booleanos para indicar si dicha casilla contiene o no una bomba y una información de estado para saber si esa casilla ha sido pisada ya o no. El constructor de la clase casilla debería recibir como argumentos, si poseerá una bomba, una referencia a la clase Tablero y las corrdenadas de dicha Casilla. El constructor ha de formar el aspecto gráfico de dicha Casilla (elementos de interfaz gráfica de usuario) y el manejador de eventos. Lo único que ha de hacer el manejador de eventos de la Casilla es llamar al método pisado de la misma.

void pisado(): Este método se encarga de mostrar la etiqueta de la casilla cuando ésta es pisada si no había sido pisada previamente. Hay que contemplar dos casos especiales. Si la casilla es una bomba, hay que llamar al método mostrarBombas de la clase Tablero. El otro caso se debe a que la casilla no tenga ninguna bomba alrededor, es decir, que el contador de bombas alrededor valga 0; en este caso, se llama al método vacío de la clase Tablero. En cualquier caso, hay que llamar al método actualizarPisadas de la clase Tablero.

void mostrarBomba(): Este método simplemente muestra la etiqueta de la celda si ésta contiene una bomba y no ha sido pisada previamente.

void etiquetar(int etiqueta): Este último método de la clase Casilla sirve para almacenar el número de bombas que existen alrededor de la Casilla y formar la etiqueta con dicho número en colores

P11

distintos dependiendo del número de que se trate. En caso de que la casilla tenga una bomba se puede poner el símbolo '@', por ejemplo.

Solución:

```
class Tablero extends Panel {
   final int dimH = 10;
   final int dimV = 10;
   final int bombas = 15;
   private int pisadas = 0;
   private Generador g;

   Casilla[][] tablero = new Casilla[dimH][dimV];

   BuscaMinas a;

   Tablero(BuscaMinas a) {
      g = new Generador(dimH*dimV,bombas);
      this.a = a;
      setLayout(new GridLayout(dimH,dimV));
      inicializar();
   }

   boolean aleatorio( int i, int j ) {
      . . .
   }

   void inicializar() {
      // inicializar con las bombas
      for(int i=0; i<dimV; i++ ) {
         for(int j=0; j<dimH; j++ ) {
            tablero[i][j] = new Casilla(aleatorio(i,j),this,i,j);
            add(tablero[i][j]);
         }
      }
      // inicializar las etiquetas
      for(int i=0; i<dimV; i++ )
         for(int j=0; j<dimH; j++ )
            tablero[i][j].etiquetar(bombasAlrededor(i,j));
   }

   int bombasAlrededor(int fila, int columna) {
      int etiqueta = 0;
      for( int i=fila-1; i<fila+2; i++ )
         for( int j=columna-1; j<columna+2; j++ )
            if( i>=0 && i<dimV && j>=0 && j<dimH && !(i==fila && j==columna) && tablero[i][j].bomba(
               etiqueta++;
            return etiqueta;
   }
```



```
   void mostrarBombas() {
      for(int i=0; i<dimV; i++)
         for(int j=0; j<dimH; j++)
            tablero[i][j].mostrarBomba();
      a.label.setText("Lo siento has perdido");
   }

   void vacio(int fila, int columna) {
      for(int i=fila-1; i<fila+2; i++)
         for(int j=columna-1; j<columna+2; j++) {
            if( i>=0 && i<dimV && j>=0 && j<dimH && !(i==fila && j==columna) )
               tablero[i][j].pisado();
         }
   }

   void actualizarPisadas() {
      if (++pisadas==dimH*dimV-bombas)
         a.label.setText("Enhorabuena! has ganado");
   }
}

class Casilla extends Panel {
   Button b;
   Label l;
   CardLayout ubi;
   Tablero t;
   private int etiqueta, fila, columna;
   private boolean bomba, pisada = false;

   Casilla(boolean bomba, Tablero t, int fila, int columna) {
      this.fila = fila;
      this.columna = columna;
      this.bomba = bomba;
      this.t = t;
      b = new Button();
      ubi = new CardLayout();
      setLayout(ubi);
      add("boton",b);

      // Gestión de eventos
      b.addActionListener( new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            pisado();
         }
      });
   }

   void mostrarEtiqueta() {
      ubi.next(this);
      pisada = true;
```

```
      t.actualizarPisadas();
   }

   void mostrarBomba() {
      if (bomba && !pisada) { ubi.next(this); }
   }

   void pisado() {
      if (!pisada) {
         mostrarEtiqueta();
         if (bomba) { l.setBackground(Color.red);
            t.mostrarBombas();
      } else {
         if (etiqueta==0)
             t.vacio(fila,columna);
       }
      }
   }

   void etiquetar(int etiqueta) {
      String s;
      Color c;
      this.etiqueta = etiqueta;
      if (bomba) { c = Color.black; s = "@"; }
      else
         switch (etiqueta) {
            case 0: { c = Color.lightGray; s = " "; break; }
            case 1: { c = Color.blue;      s = "1"; break; }
            case 2: { c = Color.green;     s = "2"; break; }
            case 3: { c = Color.red;       s = "3"; break; }
            case 4: { c = Color.black;     s = "4"; break; }
            case 5: { c = Color.pink;      s = "5"; break; }
            default:{ c = Color.black;     s = ""+etiqueta; break; }
         }
      l = new Label(s,Label.CENTER);
      l.setForeground(c);
      l.setBackground(Color.lightGray);
      add("etiqueta",l);
   }

   boolean bomba() { return bomba; }
}
```

Dispositivos Leer Escribir Cifrar/comprimir

Problema 2 (2.25 ptos)

Una cuestión muy importante en la computación es el tratamiento de la entrada/salida de datos puesto que es el mecanismo de comunicación de los programas con el exterior. En nuestro caso vamos a simular estos flujos de datos.

En general, todo flujo de datos se puede leer o escribir desde o sobre un dispositivo. Por esto los dispositivos podran ser de lectura, de escritura o ambos. Además sobre algunos de ellos se permite que el flujo de datos fluya cifrado y/o comprimido.

Como se deduce, las operaciones básicas son la de lectura y escritura. Es obvio que cada dispositivo lee y escribe de forma diferente; además, en función que el flujo sea cifrado o comprimido la lectura y escritura deberá cambiar, puesto que el proceso de cifrado y compresión se incluye en el propio método de leer y escribir. Es muy importante que nos demos cuenta que estos dispositivos SÓLO leen o escriben, es decir, no debemos pensar que el hecho de que, por ejemplo, se lea de forma cifrada implique que es un método con nombre diferente. Es el método de lectura que tiene diferente codificación.

En nuestro caso los dispositivos a tratar serán discos, terminales, impresoras y puertos de red (sockets). Como sabemos, algunos de estos dispositivos son de lectura, de escritura o ambos. De éstos, los discos y los sockets pueden ser cifrados, comprimidos o ambos.

Se trata de escribir una relación de clases en las que según los datos anteriores sea muy fácil hacer cuestiones como las siguientes:

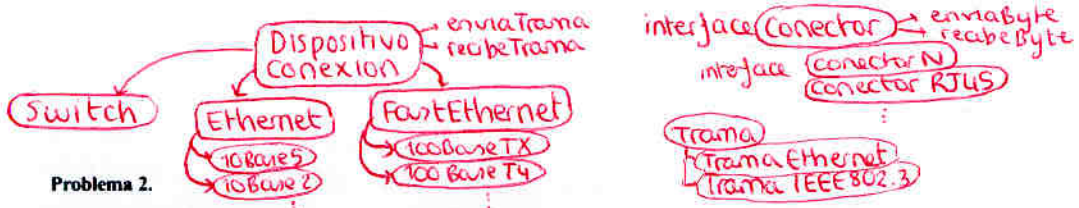
- Tener una estructura de datos (lista, vector, etc..) sobre la que pueda almacenar cualquier dispositivo.
- Tener las estructuras de datos (lista, vector, etc..) sobre las que pueda almacenar cualquier dispositivo de entrada, de salida o de entrada/salida separadamente.
- Permitir métodos que puedan aceptar como parámetros cualquier dispositivo que sea capaz de escribir, por ejemplo, aunque lo haga de formar comprimida o cifrada.

Solución:

```
class FlujoDatos {
}

interface Leer {
   void leer();
}

interface Escribir {
   void escribir();
}

interface LeerEscribir extends Leer, Escribir{
   void leer();
   void escribir();
}

interface Cifrar extends LeerEscribir {
   void leer();
   void escribir();
}

interface Comprimir extends LeerEscribir {
   void leer();
   void escribir();
}

class Disco extends FlujoDatos implements LeerEscribir{
```


P12

```
    public void leer(){};
    public void escribir () {};
}

class DiscoCifrado extends Disco implements Cifrar {
    public void leer(){};
    public void escribir () {};
}

class DiscoComprimido extends Disco implements Comprimir{
    public void leer(){};
    public void escribir () {};
}

class DiscoCifradoComprimido extends Disco implements Cifrar, Comprimir {
    public void leer(){};
    public void escribir () {};
}

class Terminal extends FlujoDatos implements Escribir{
    public void escribir() {};
}

class Impresora extends FlujoDatos implements Escribir{
    public void escribir() {};
}

class Socket extends FlujoDatos implements LeerEscribir{
    public void leer(){};
    public void escribir () {};
}

class SocketCifrado extends Socket implements Cifrar {
    public void leer(){};
    public void escribir () {};
}

class SocketComprimido extends Socket implements Comprimir{
    public void leer(){};
    public void escribir () {};
}

class SocketCifradoComprimido extends Socket implements Cifrar, Comprimir {
    public void leer(){};
    public void escribir () {};
}
```



Problema 2.

Se pretende estudiar el rendimiento de redes de área local (LAN's) mediante la implementación de un programa de simulación en Java. Para ello, se describen a continuación las características que tienen los dispositivos conectados a esta red.

La red estará formada por dispositivos de tipo Ethernet (IEEE802.3) que transmiten a 10Mbps y Fast Ethernet (IEEE802.3u) que transmiten a 100Mbps. El método de envío y recepción de tramas varía según sean de un tipo u otro. La especificación define cuatro tipos en la subcapa física para los dispositivos Ethernet con las siguientes características básicas:

- 10Base5: Cable coaxial grueso de un par y conector tipo N,
- 10Base2: Cable coaxial delgado de un par y conector tipo BNC,
- 10BaseT: Cable UTP de 2/2 pares y conector tipo RJ45,
- 10BaseFL: Cable de fibra óptica de dos pares con conector ST.

Los dos últimos tipos admiten comunicación full-duplex. Para los dispositivos de tipo FasEthernet se dispone de tres especificaciones en la subcapa física:

- 100BaseTX: Cable UTP de dos pares con conector RJ-45.
- 100BaseT4: Dos cables UTP de dos pares con conector RJ-45.
- 100BaseFX: Dos hilos de fibra conector SC.

Sólo admiten transmisión de tipo full-dúplex el primero y el tercero de los dos dispositivos anteriores. Cualquier dispositivo puede conectarse a cualquier otro mientras el tipo de conector y la velocidad de transmisión coincidan. Cada conector define sus propias características eléctricas (niveles de tensión, cronogramas, etc.). El/la envío/recepción de una trama se divide en diversos envíos/recepciones de bytes los cuales dependen del tipo de conector.

Aunque en esta primera simulación no aparecen, se pretende implementar el sistema de manera que más adelante puedan incorporarse otras especificaciones como Gigabit Ethernet (IEEE802.3z: 1000Base-CX, 1000Base-SX o 1000Base-LX) o Ethernet a 10Gigabits (IEEE 802.3ae).

En la capa de acceso al medio (MAC) todos los dispositivos se caracterizan por disponer de una dirección MAC compuesta por 6 bytes. Los dispositivos env reciben tramas con el formato descrito en la especificación. La especificación describe dos tipos de trama: la Ethernet y la IEEE802.3.

El tipo de red que se desea modelizar es conmutada, es decir, posee switches multipuerta. Estos dispositivos tienen un cierto número de puertas todas con capacidad tanto para transmisión a 10 como 100Mbps al que pueden conectarse los dispositivos. Las puertas se diferencian entre sí por el tipo de conector (N, BNC, RJ45, etc.) que definen las características físicas eléctricas del canal de transmisión. Entre sus características figuran, además del número de puertas para cada tipo de conector, el número máximo de direcciones MAC que pueden soportar. A una puerta puede conectarse también otro switch. Los switches poseen dos modos de funcionamiento: el modo normal o Store and Forward en el cual realizan comprobación del CRC de cada trama y el modo Cut-Through en el que no lo hacen.

y para ello requiere una conexión conmutada entre los nodos origen y destino, que es provista por los switches LAN. Esta conexión es considerada punto a punto y libre de colisiones, por lo tanto, para que Ethernet Full Dúplex funcione deben darse ciertas condiciones. El switch realiza esta comprobación cada vez que recibe una trama y la

La introducción del funcionamiento Full Dúplex introduce una nueva funcionalidad, el control de flujo, que se implementa mediante el comando PAUSE. El receptor puede en cualquier momento enviar al emisor un comando PAUSE indicándole por cuanto tiempo debe dejar de enviarle datos.

Se pide:

Determinar las diferentes clases, clases abstractas e interfaces necesarias para la modelización de las componentes mencionadas, estableciendo la estructura jerárquica que las relaciona, sus atributos y métodos, de estos últimos indicar únicamente el nombre y si son o no abstractos. **.(2 puntos)**

SOLUCIÓN

```
abstract class Dispositivo_de_conexion {
      byte MAC[] = new byte[6];
      abstract void enviar( Trama t);
      abstract Trama recibir( );
}

class Ethernet extends Dispositivo_de_conexion {
      void enviar( Trama t) { }
      Trama recibir( ) { }
}

class FastEthernet extends Dispositivo_de_conexion {
      void enviar( Trama t) { }
      Trama recibir( ) { }
}

class _10Base5 extends Ethernet implements ConectorN {
  public void enviaByte( byte b ) { }
  public byte recibeByte( ) { }
}

class _10Base2 extends Ethernet implements ConectorBNC {
  public void enviaByte( byte b ) { }
  public byte recibeByte( ) { }
}

class _10BaseT extends Ethernet implements ConectorRJ45, FullDuplex {
  public void enviaByte( byte b ) { }
  public byte recibeByte( ) { }
      public void PAUSE() { }
}

class _10BaseFL extends Ethernet implements ConectorST, FullDuplex {
  public void enviaByte( byte b ) { }
  public byte recibeByte( ) { }
```

```
      public void PAUSE() { }
}

class _100BaseTX extends FastEthernet implements ConectorRJ45,
FullDuplex {
  public void enviaByte( byte b ) { }
  public byte recibeByte( ) { }
      public void PAUSE() { }
}

class _100BaseT4 extends FastEthernet implements ConectorRJ45 {
  public void enviaByte( byte b ) { }
  public byte recibeByte( ) { }
}

class _100BaseFX extends FastEthernet implements ConectorSC,
FullDuplex {
  public void enviaByte( byte b ) { }
  public byte recibeByte( ) { }
      public void PAUSE() { }
}

interface Conector {
      /*Caracteristicas eléctricas (constantes)*/
  void enviaByte( byte b );
  byte recibeByte( );
}

interface ConectorN extends Conector {
      /*Caracteristicas eléctricas (constantes)*/
}
interface ConectorBNC extends Conector {
      /*Caracteristicas eléctricas (constantes)*/
}
interface ConectorRJ45 extends Conector {
      /*Caracteristicas eléctricas (constantes)*/
}
interface ConectorST extends Conector {
      /*Caracteristicas eléctricas (constantes)*/
}
interface ConectorSC extends Conector {
      /*Caracteristicas eléctricas (constantes)*/
}

interface FullDuplex {
      void PAUSE();
}

abstract class Trama {
      // Formato de la trama
      byte MAC[] = new byte[6]; //Dirección de destino
}

class TramaEthernet extends Trama { }

class TramaIEEE extends Trama { }

class Switch extends Dispositivo_de_conexion {
  ConectorN            CN[];
  ConectorBNC   CBNC[];
  ConectorRJ45  CRJ45[];
  ConectorST           CST[];
  ConectorSC           CSC[];
      int MaxMAC;
      boolean ModoDeFunc;

      Switch( /*...*/ ) {     /*...*/ }

      void enviar( Trama t, Dispositivo_de_conexion ds ) { }

      Trama recibir( Dispositivo_de_conexion or ) { }
}


class Problema2 {
}
```