



Fundamentos de Telemática.
Área de Ingeniería Telemática
Departamento de Ingeniería de Sistemas y Automática
Escuela Superior de Ingenieros
Universidad de Sevilla



Fundamentos de Telemática 3º Ingeniero de Telecomunicación

EXÁMENES DE CURSOS ANTERIORES

Ejercicio 1 (25/01/2001) E/S, excepciones.

El objetivo de este ejercicio es escribir un programa en Java que copie el contenido de un fichero en otro fichero con un nombre diferente. La llamada al intérprete deberá ser de la siguiente forma:

java Ejercicio1 origen.txt destino.txt

1. ¿Qué paquete necesita importar?
2. Mediante el uso de las clases **FileInputStream** y **FileOutputStream**, realice la **declaración, instanciación** y utilice el **constructor** apropiado para realizarlo de acuerdo con la llamada al intérprete que aparece arriba. Los constructores de las clases aparecen en la parte inferior de esta hoja. **Nota: IndexOutOfBoundsException** esta excepción se produce cuando algún tipo de índice está fuera de rango.
3. Utilice los métodos de las clases **FileInputStream** y **FileOutputStream** que aparecen a continuación, para realizar la operación de lectura/escritura **byte a byte**. **Nota:** El método **int read()** devuelve el byte leído o -1 si ha alcanzado el fin de fichero.
4. Cierre todos los flujos.
5. El método **available()** devuelve el número de bytes que pueden leerse sin bloqueo. Úselo para leer/escribir todos los datos de una vez. Utilice una tabla de bytes. **Nota: int read(byte[] b)** y **int read(byte[] b, int off, int len)** devuelve el número de caracteres leídos o -1 si ha alcanzado fin de fichero.
6. Realice la operación de lectura/escritura manejando bloques de 512 bytes.

Algunas de las operaciones las realizamos con las clases **DataInputStream** y **DataOutputStream**. (Los métodos de estas clases se proporcionan en este enunciado).

7. Realice la **declaración, instanciación y construcción**, utilizando los constructores para crear objetos de las clases **DataInputStream** y **DataOutputStream**. Obtenga las referencias **InputStream** y **OutputStream** de los constructores, con los constructores del apartado 2.

<i>Los constructores de la clase DataInputStream</i>	<i>Los constructores de la clase DataOutputStream</i>
DataInputStream (InputStream in) throws IOException	DataOutputStream (OutputStream out) throws IOException

8. Realice la lectura/escritura de los datos byte a byte

<i>Los constructores de la clase FileInputStream</i>	<i>Los constructores de la clase FileOutputStream</i>
FileInputStream (String archivo) throws FileNotFoundException FileInputStream (FileDescriptor fdObj) throws FileNotFoundException FileInputStream (File file) throws FileNotFoundException	FileOutputStream (String nombre) throws IOException FileOutputStream (File file) throws IOException FileOutputStream (FileDescriptor fd) throws IOException FileOutputStream (String nombre, boolean append) throws IOException

<i>Los métodos de la Clase FileInputStream</i>	<i>Los métodos de la Clase FileOutputStream</i>
int read() throws IOException int read(byte[] b) throws IOException int read(byte[] b, int off, int len) throws IOException int available() void close() throws IOException long skip(long n)	void write(int b) throws IOException void write(byte[] b) throws IOException void write(byte[] b, int off, int len) throws IOException void close() throws IOException

Solución del Ejercicio1 (apartados del 1 al 6)

```
/*
 * Ejercicio1.java
 * Solución al Ejercicio 1 del examen de Enero de 2001 de
 * Fundamentos de Telemática.
 */

import java.io.*; //Apartado 1

/** Clase Ejercicio1 que contiene al metodo main
 */
public class Ejercicio1{

    /** Comprueba los argumentos en línea de comandos
     * @param args Argumentos de la línea de comandos
     */
    public static void main(String args []){
        FileInputStream in = null; //Apartado 2
        FileOutputStream out = null;

        try{
            in = new FileInputStream(args[0]);
            out = new FileOutputStream(args[1]);
        }catch(FileNotFoundException e){
            System.out.println("Fichero de entrada no encontrado." +e);
        }catch(IOException e){
            System.out.println(e);
        }catch(IndexOutOfBoundsException e){
            System.out.println("Llamada al interprete: java Ejercicio1
                                origen.txt destino.txt "+e);
        }
    }

    /**
     * try{ //Apartado 3
     *     while((c = in.read())!=-1)
     *         out.write(c);
     *     }catch(IOException e){
     *         System.out.println(e);
     *     }
     */

    /**
     * try{ //Apartado 5
     *     byte[] b = new byte[in.available()] ;
     *     in.read(b);
     *     out.write(b);
     * }catch(IOException e){
     *     System.out.println(e);
     * }
     */

    /**
     * try{ //Apartado 6
     *     byte[] b = new byte [512];
     *     while((c = in.read(b,0,512))!=-1)
     *         out.write(b,0,c);
     * }catch(IOException e){
     *     System.out.println(e);
     * }
     */
}
```

```

/*          try{                                //Apartado 4
*          in.close();
*          out.close();
*          }catch(IOException e){
*          System.out.println(e);
*          }
*/
}
}

```

Solucion del Ejercicio1 (apartados 7 y 8)

```

import java.io.*;

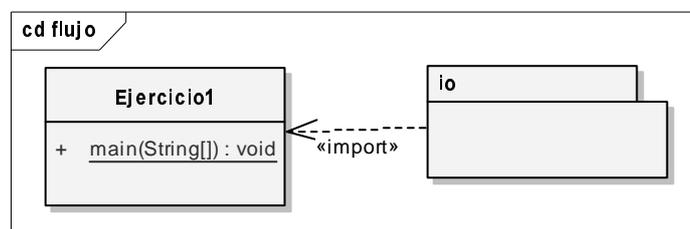
/** Creamos objetos de las clases DataInputStream y DataOutputStream
 * y copiamos byte a byte.
 * La clase Ejerciciolbis contiene el metodo main
 */
public class Ejerciciolbis{
    public static void main(String args[]){
        DataInputStream in = null;                                //Apartado 7
        DataOutputStream out = null;

        try{
            in = new DataInputStream(new FileInputStream(args[0]));
            out = new DataOutputStream(new FileOutputStream(args[1]));
        }catch(FileNotFoundException e){
            System.err.println(e);
            return;
        }catch(IOException e){
            System.err.println(e);
            System.exit(1);
        }catch(IndexOutOfBoundsException e){
            System.err.println(e);
            System.exit(1);
        }

        int c;
        try{                                                       //Apartado 8
            byte[] b = new byte[512];
            while((c = in.read(b))!=-1)
                out.write(b);
        }catch(IOException e){
            System.err.println(e);
        }

        try{
            in.close();
            out.close();
        }catch(IOException e){
            System.out.println(e);
        }
    }
}

```



Class Ejercicio1

java.lang.Object

Ejercicio1

```
public class Ejercicio1
extends java.lang.Object
```

La clase Ejercicio1 contiene los métodos main y calcula

Constructor Summary

<u>Ejercicio1</u> ()	
----------------------	--

Method Summary

static void	<u>calcula</u> (int a) Lanza la excepción si el valor del entero está fuera del rango [0,10]
static void	<u>main</u> (java.lang.String[] args) Llama a la función calcula

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ejercicio1

```
public Ejercicio1()
```

Method Detail

calcula

```
public static void calcula(int a)
    throws MiExcepcion
    Lanza la excepción si el valor del entero está fuera del rango [0,10]
Throws:
    MiExcepcion
```

main

```
public static void main(java.lang.String[] args)
    Llama a la función calcula
```

Ejercicio 2 (25/01/2001) socket, E/S, excepciones

En este ejercicio se codifica un programa en Java que implementa la parte correspondiente al cliente para utilizar el servicio de eco.

Un socket es un punto de comunicación bidireccional entre programas funcionando en lugares distintos en la red. La comunicación bidireccional se implementa en Java utilizando los flujos de entrada y salida (**InputStream** y **OutputStream**) por lo que se consigue manejar de manera uniforme el acceso a ficheros y a conexiones remotas.

El **servicio de eco** consiste en un servidor que acepta conexiones, y todo lo que recibe por su entrada lo replica por su salida. Así, un cliente le envía datos y recibe los mismos. En las máquinas UNIX el servidor de eco está en el puerto número 7.

1. ¿Qué paquetes necesita importar?
2. Realice la apertura de un socket (haga la **declaración, instanciación y construcción** de un objeto **Socket**), que se conectará a una máquina de nombre **adriano**, al puerto número **7**. Utilizando el constructor: **Socket (String host, int puerto) throws IOException, UnknownHostException**.
3. Para la creación del flujo de entrada para recibir las respuestas del servidor se utilizará la clase **DataInputStream**. Esta clase tiene un único constructor: **DataInputStream (InputStream in)**. Use el método **getInputStream()** de la clase **Socket** para obtener el flujo de entrada asociado al socket.
InputStream getInputStream() throws IOException
4. Para la creación del flujo de salida para enviar información al servidor se utilizará la clase **DataOutputStream**. Esta clase tiene un único constructor: **DataOutputStream (OutputStream out)**. Use el método **getOutputStream()**, de la clase **Socket** para obtener el flujo de salida asociado al socket.
OutputStream getOutputStream() throws IOException
5. Envíe a través del flujo de salida la cadena "Hola\n", y recójalo con el método **readLine()** y sáquelo por pantalla.
6. Cierre todos los flujos.

El método **readLine()** ha sido desaprobado (*deprecated*) en las nuevas versiones del compilador. Para poder utilizar este método realice los cambios que sugiere en el método.

7. Realice las modificaciones que sugiere el método **readLine()** de la clase **DataInputStream**.
8. ¿Cómo se debería instanciar, para leer desde la entrada estándar?

<i>Los métodos de la clase DataInputStream</i>	
int read() throws IOException	Lee un byte del flujo de entrada.
int read(byte[] b) throws IOException	Lee caracteres del flujo de entrada y los pone en la matriz b.
int read(byte[] b, int off, int len) throws IOException	Lee len bytes del flujo de entrada y lo pone en la matriz b.
boolean readBoolean() throws IOException	Lee un byte de la entrada y devuelve true si es distinto de cero, false si es 0.
byte readByte() throws IOException	Lee y devuelve un byte de la entrada.
char readChar() throws IOException	Lee y devuelve un char de la entrada.
double readDouble() throws IOException	Lee 8 bytes de entrada y devuelve un double.
float readFloat() throws IOException	Lee 4 bytes de entrada y devuelve un float.
void readFully(byte[] b) throws IOException	Lee bytes de la entrada y los almacena en el buffer b.
void readFully(byte[] b, int off, int len) throws IOException	Lee len bytes de la entrada.
int readInt() throws IOException	Lee 4 bytes de entrada y devuelve un entero.
String readLine() throws IOException	Lee la siguiente línea de texto de la entrada. Deprecated. Este método no convierte de forma apropiada bytes a caracteres. La forma más apropiada para leer una línea de texto es con el método BufferedReader.readLine() . Los programas que usan la clase DataInputStream para leer líneas pueden convertirlo usando la clase BufferedReader reemplazando el código de la siguiente forma: DataInputStream d = new DataInputStream (in); con
long readLong() throws IOException	Lee 8 bytes de entrada y devuelve un long.
short readShort() throws IOException	Lee 2 bytes de entrada y devuelve un short.
int readUnsignedByte() throws IOException	Lee 1 bytes de entrada, devuelve un entero en el rango de 0 a
int readUnsignedShort() throws IOException	Lee 2 bytes de entrada y devuelve un entero en el rango de 0 a 65535.
int skipBytes(int n) throws IOException	Descarta los n bytes de datos del flujo de entrada.

<i>Los métodos de la clase DataOutputStream</i>	
void flush() throws IOException	Limpia el flujo de datos de salida.
int size()	Devuelve el valor del número de bytes escritos en el flujo de salida de datos.
void write(byte[] b, int off, int len) throws IOException	Escribe len bytes de la matriz b, en el flujo de salida.
void write(int b) throws IOException	Escribe en el flujo de salida un byte.
void writeBoolean(boolean v) throws IOException	Escribe un valor de tipo boolean en el flujo de salida.
void writeByte(int v) throws IOException	Escribe en el flujo de salida los 8 bits de menor peso del argumento
void writeBytes(String s) throws IOException	Escribe una cadena en el flujo de salida.
void writeChar(int v) throws IOException	Escribe un valor char, en el flujo de salida.
void writeChars(String s) throws IOException	Escribe cada carácter (dos bytes) en la cadena s, en el flujo de
void writeDouble(double v) throws IOException	Escribe un valor double (8 bytes), en el flujo de salida.
void writeFloat(float v) throws IOException	Escribe un valor float (4 bytes), en el flujo de salida.
void writeInt(int v) throws IOException	Escribe un valor int (4 bytes), en el flujo de salida.
void writeLong(long v) throws IOException	Escribe un valor long, en el flujo de salida.
void writeShort(int v) throws IOException	Escribe un valor short, en el flujo de salida.

Solución del Ejercicio 2 (apartados del 1 al 6)

```
/*
 * EcoCliente.java
 *
 * Solución al Ejercicio 2 del examen de Enero de 2001 de
 * Fundamentos de Telemática.
 */

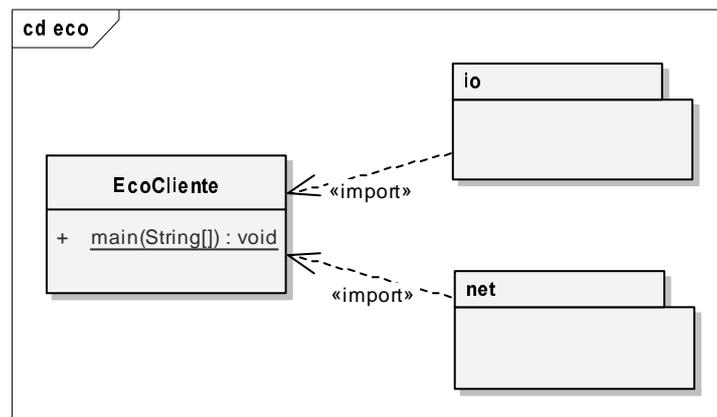
import java.io.*; //Apartado 1
import java.net.*;

/** Servicio eco con el metodo desaprobadado */
public class EcoCliente {
    public static void main(String[] args){
        Socket s = null; //Apartados 2,3 y 4
        DataInputStream in = null;
        DataOutputStream out = null;

        try {
            s = new Socket("adriano", 7) ;
            in = new DataInputStream(s.getInputStream());
            out = new DataOutputStream(s.getOutputStream());
        } catch (IOException e) {
            System.err.println(e);
            System.exit(1);
        }

        try{ //Apartado 5
            out.writeChars(new String ("Hola\n"));
            System.out.println(in.readLine());
        }catch(IOException e){
            System.err.println(e) ;
        }

        try{ //Apartado 6
            out.close();
            in.close();
            s.close();
        }catch(IOException e){
            System.err.println(e);
        }
    }
}
```



Class EcoCliente

java.lang.Object

EcoCliente

```
public class EcoCliente
extends java.lang.Object
```

Servicio eco con el metodo desaprobado

Constructor Summary

<u>EcoCliente</u> ()	
-----------------------------	--

Method Summary

static void <u>main</u> (java.lang.String[] args)	
--	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

EcoCliente

```
public EcoCliente()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Solución del Ejercicio 2 (apartados del 7 y 8)

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

/** Servicio eco con el metodo actualizado */
public class EcoCliente {
    public static void main(String[] args){
        Socket s = null;
        BufferedReader in = null;
        DataOutputStream out = null;

        try {
            //Apartados 7 y 8
            s = new Socket("adriano", 7) ;
            in = new BufferedReader(new
                InputStreamReader(s.getInputStream()));
            out = new DataOutputStream(s.getOutputStream());
        } catch (IOException e) {
            System.err.println(e);
            System.exit(1);
        }

        try{
            out.writeChars(new String ("Hola\n"));
            System.out.println(in.readLine());
        }catch(IOException e){
            System.err.println(e) ;
        }

        try{
            out.close();
            in.close();
            s.close();
        }catch(IOException e){
            System.err.println(e);
        }
    }
}
```

Ejercicio 3 (25/01/2001) herencia, interfaz, excepciones, hilos, applet, awt.

En este ejercicio se implementa un applet con un cartel moviéndose. El cartel va a estar en un String: `String msg = " un cartel moviendose "`; y se va a realizar mediante el uso de hilos `Thread t = null`; implementando la interfaz `Runnable`. La suspensión y reanudación del hilo se realizará con los métodos `resume` y `suspend` (`final void resume()` y `final void suspend()`). Ambas, `msg` y `t` serán variables de instancia.

Cuando la applet necesite actualizar la información mostrada en su ventana, se llamará al método `void repaint()`, de modo que cuando se realice la actualización del cartel se realizará llamando a este método. La etiqueta HTML que llama al applet es

```
<applet code="Ejercicio3" width=300  
height=50> </applet>
```

1. ¿Qué paquetes necesita importar?
2. Dé nombre a la clase que implementa la applet, especificando si hereda o implementa de alguna clase o interfaz.
3. Implemente el primer método que ejecuta la applet, `public void init()`, realice la instanciación del hilo y de comienzo a su ejecución del hilo y suspéndalo inmediatamente después.
4. Implemente el método de *inicialización* de la applet que permite reiniciar una applet después de que haya sido parada `public void start()`. Este método solamente reanudará el hilo.
5. Implemente el tercero de los métodos de *inicialización* el método `public void paint(Graphics g)`. Este método muestra el cartel (String) con el método `void drawString(String mensaje, int x, int y)`, en la posición 50, 30.
6. Implemente el primero de los métodos de *finalización* `public void stop()`. Este método solamente suspenderá temporalmente el hilo.
7. Implemente el segundo de los métodos de *finalización* `public void destroy()`. Este método solamente destruirá el hilo con el método `stop()` de la clase `Thread`.
8. Implemente el método `public void run()` que establece el punto de entrada al hilo. En este método se desplazan hacia la izquierda de manera repetida los caracteres de la cadena `msg`. El hilo se "dormirá" durante 250 milisegundos.

`char charAt(int index)` devuelve el carácter `index` de una cadena.

`String substring(int start, int end)` devuelve la subcadena especificada.

Solución del Ejercicio 3

```
/*
<applet code="Ejercicio3" width=300 height=50>
</applet>
*/

import java.awt.*; //Apartado 1
import java.applet.*;

/** Clase AnimaApplet1 hereda de Applet e implementa Runnable
 */
public class Ejercicio3 extends Applet implements Runnable{ //Apartado 2
    String msg = " un cartel moviendose ";
    Thread t = null;

    /** Establece los colores e inicia el hilo */
    public void init(){ //Apartado 3
        setBackground(Color.cyan);
        setForeground(Color.red);
        t = new Thread(this);
        t.start(); //comienza la ejecución del hilo
        t.suspend(); //suspende la applet hasta que este
                    //totalmente inicializada
    }

    /** Reanuda el hilo.*/
    public void start(){ //Apartado 4
        t.resume();
    }

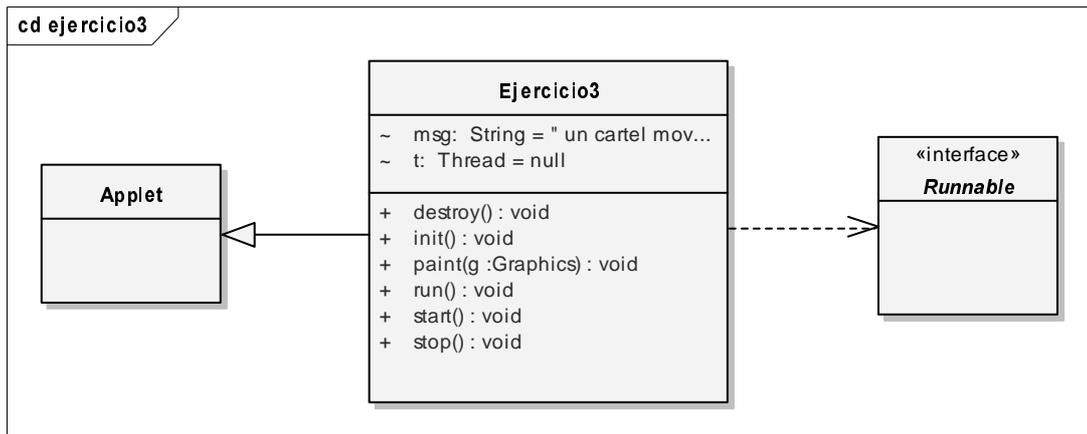
    public void run(){ //Apartado 8
        char ch;

        //muestra el cartel hasta que se para.
        for(;;){
            try{
                repaint();
                Thread.sleep(250);
                ch = msg.charAt(0);
                msg = msg.substring(1,msg.length());
                msg += ch;
            }catch(InterruptedException e){}
        }
    }

    /** Para el cartel */
    public void stop(){ //Apartado 6
        t.suspend();
    }

    /** Para el hilo cuando la applet termina */
    public void destroy(){ //Apartado 7
        if(t!=null){
            t.stop ();
            t = null;
        }
    }

    /** Muestra el cartel */
    public void paint (Graphics g) { //Apartado 5
        g.drawString (msg, 50,30);
    }
}
```



[Package](#) **[Class](#)** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Class Ejercicio3

```

java.lang.Object
├ java.awt.Component
│   └ java.awt.Container
│       └ java.awt.Panel
│           └ java.applet.Applet
│               └ Ejercicio3
  
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.lang.Runnable, javax.accessibility.Accessible

```

public class Ejercicio3
extends java.applet.Applet
implements java.lang.Runnable
  
```

Solución al Ejercicio 3 del examen de Enero de 2001 de Fundamentos de Telemática.

See Also:

[Serialized Form](#)

Nested Class Summary

Nested classes/interfaces inherited from class java.applet.Applet

java.applet.Applet.AccessibleApplet

Nested classes/interfaces inherited from class java.awt.Panel

java.awt.Panel.AccessibleAWTPanel

Nested classes/interfaces inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

Nested classes/interfaces inherited from class java.awt.Component

java.awt.Component.AccessibleAWTComponent,
java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT,
TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Ejercicio3()

Method Summary

void	<u>destroy()</u> Para el hilo cuando la applet termina
void	<u>init()</u> Establece los colores e inicia el hilo
void	<u>paint</u> (java.awt.Graphics g) Muestra el cartel
void	<u>run()</u>
void	<u>start()</u> Reanuda el hilo.
void	<u>stop()</u> Para el cartel

Methods inherited from class java.applet.Applet

getAccessibleContext, getAppletContext, getAppletInfo, getAudioClip, getAudioClip, getCodeBase, getDocumentBase, getImage, getImage, getLocale, getParameter, getParameterInfo, isActive, newAudioClip, play, play, resize, resize, setStub, showStatus

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Ejercicio3

```
public Ejercicio3()
```

Method Detail

init

```
public void init()  
    Establece los colores e inicia el hilo  
Overrides:  
    init in class java.applet.Applet
```

start

```
public void start()  
    Reanuda el hilo.  
Overrides:  
    start in class java.applet.Applet
```

run

```
public void run()  
Specified by:  
    run in interface java.lang.Runnable
```

stop

```
public void stop()  
    Para el cartel  
Overrides:  
    stop in class java.applet.Applet
```

destroy

public void **destroy**()

Para el hilo cuando la applet termina

Overrides:

destroy in class `java.applet.Applet`

paint

public void **paint**(`java.awt.Graphics g`)

Muestra el cartel

Overrides:

paint in class `java.awt.Container`

[Package](#) **[Class](#)** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

(16/01/2001 y 1/06/2004)
paquetes.

polimorfismo, herencia, interfaces y

Para la realización de los ejercicios 1 y 2 hay hojas adicionales. En estas hojas aparece el nombre de la clase de la que tratan los ejercicios (**Qname**), el paquete al que pertenece, lo que implementa y lo que hereda, los constructores de la clase, y los métodos que contiene. El objetivo del primer ejercicio es el de implementar el código en Java completo de la clase y el del segundo ejercicio el diagrama UML de la clase.

Ejercicio 1

La clase **QName** representa un “*nombre cualificado*” en la forma en la que está especificada en las normas de *w3c (XML Schema Part2: Datatypes specification)*. El nombre cualificado lo vamos a representar con dos partes que son el espacio de nombres y la parte local. Un objeto de la clase **QName** deberá contener, por tanto, la información relacionada con el espacio de nombres representado por **namespaceURI** (esta parte podrá no estar presente) y la parte local contenida en **localPart**.

1. Especifique nombre de la clase, visibilidad, indicando lo que hereda y lo que implementa (si corresponde) y paquete en el que está implementado. Véase hojas anexas.
2. Implemente los constructores que tiene la clase (uno con un parámetro **String** y otro dos).
3. Implemente los métodos “**get**” (**getNamespaceURI()** y **getLocalPart()**).
4. Implemente el método **toString()**, que devuelva como **String** el objeto **Qname** en forma siguiente “**{namespaceURI}localPart**”, con la parte “**{namespaceURI}**” opcional.
5. Implemente el método “inverso” a **toString()**, **valueOf()**, que devuelva como **QName** la representación de un objeto **String**.
6. Implemente el método **equals**, que devuelva cierto si el objeto es instancia de **QName**, (la expresión (x **instanceof Q**) devuelve cierto (**true**) si la referencia x es instancia de la clase **Q**), y además coinciden tanto la “parte local” como el “espacio de nombres”.
7. Un código hash es una forma de calcular una “clave” para asociarla a un objeto. Esta clave se puede obtener invocando al método **hashCode()** desde la referencia a un **String**, si es distinta de **null**. Implemente el método **hashCode()** de la clase **QName**, mediante la suma de la invocación a los objetos de que consta la clase **QName** (deberá ser **0** si ambas son **null**).

Nota: Algunos métodos de la clase **String**, que pueden ser de ayuda:

<i>Métodos</i>	<i>Breve descripción</i>
Char charat(int index)	Devuelve el carácter en la posición que especifica el índice.
int indexOf(String str) int indexOf(char ch) int indexOf(char ch, int fromIndex) int indexOf(String str, int fromIndex)	Devuelve el índice de la primera ocurrencia del String , o char a partir de una posición, según la versión del método.
String substring(int beginIndex) String substring(int beginIndex, int endIndex)	Devuelve un Nuevo String generado a partir de otro.

Ejercicio 2

Dibuje de forma detallada el diagrama de clases UML de **QName**.

javax.xml.namespace
Class QName

java.lang.Object
↳ [javax.xml.namespace.QName](#)
All Implemented Interfaces:
java.io.Serializable

public class **QName**
extends java.lang.Object
implements java.io.Serializable

QName class represents the value of a qualified name as specified in [XML Schema Part2: Datatypes specification](#).

The value of a QName contains a **namespaceURI**, a **localPart** and a **prefix**. The localPart provides the local part of the qualified name. The namespaceURI is a URI reference identifying the namespace.

Version:
1.0

See Also:
[Serialized Form](#)

Constructor Summary

[QName](#)(java.lang.String localPart)
Constructor for the QName.
[QName](#)(java.lang.String namespaceURI, java.lang.String localPart)

Constructor for the QName.

Method Summary

boolean	equals (java.lang.Object obj) Tests this QName for equality with another object.
java.lang.String	getLocalPart () Gets the local part for this QName.
java.lang.String	getNamespaceURI () Gets the namespace URI for this QName.
int	hashCode () Returns a hash code value for this QName object.
java.lang.String	toString () Returns a string representation of this QName.
static QName	valueOf (java.lang.String s) Returns a QName holding the value of the specified String.

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait

Constructor Detail

QName

public **QName**(java.lang.String localPart)
Constructor for the QName.

Parameters:

localPart - local part of the QName

QName

```
public QName(java.lang.String namespaceURI,  
             java.lang.String localPart)
```

Constructor for the QName.

Parameters:

namespaceURI - namespace URI for the QName
localPart - local part of the QName.

Method Detail

getNamespaceURI

```
public java.lang.String getNamespaceURI()  
Gets the namespace URI for this QName.
```

Returns:

namespace URI

getLocalPart

```
public java.lang.String getLocalPart()  
Gets the local part for this QName.
```

Returns:

the local part

toString

```
public java.lang.String toString()  
Returns a string representation of this QName.
```

Returns:

a string representation of the QName

equals

```
public boolean equals(java.lang.Object obj)
```

Tests this QName for equality with another object.

If the given object is not a QName or is null then this method returns false.

For two QNames to be considered equal requires that both localPart and namespaceURI must be equal. This method uses String.equals to check equality of localPart and namespaceURI. Any class that extends QName is required to satisfy this equality contract.

This method satisfies the general contract of the Object.equals method.

Parameters:

obj - the reference object with which to compare

Returns:

true if the given object is identical to this QName; false otherwise.

valueOf

```
public static QName valueOf(java.lang.String s)  
Returns a QName holding the value of the specified String.
```

The string must be in the form returned by the QName.toString() method, i.e. "{namespaceURI}localPart", with the "{namespaceURI}" part being optional.

This method doesn't do a full validation of the resulting QName. In particular, it doesn't check that the resulting namespace URI is a legal URI (per RFC 2396 and RFC 2732), nor that the resulting local part is a legal NCName per the XML Namespaces specification.

Parameters:

s - the string to be parsed

Returns:

QName corresponding to the given String

Throws:

java.lang.IllegalArgumentException - If the specified String cannot be parsed as a QName

hashCode

```
public int hashCode()
```

Returns a hash code value for this QName object. The hash code is based on both the localPart and namespaceURI parts of the QName. This method satisfies the general contract of the Object.hashCode method.

Returns:

a hash code value for this QName object

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#) [All Classes](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) | [DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

Copyright © 2003 Apache Web Services Project. All Rights Reserved.

Solución del Ejercicio 1

```
/*
 * QName.java
 *
 * Solución al Ejercicio 1 del examen de Junio de 2004 de
 * Fundamentos de Telemática.
 */

package javax.xml.namespace;

/** Representa un "nombre cualificado" como es especificado en XML Schema
 * Part2: Datatypes specification.
 * Contiene un namespaceURI, espacio de nombres,
 * y un localPart, parte local.
 */
public class QName implements java.io.Serializable{

    private String namespaceURI;
    private String localPart;

    /** Constructor con un parámetro
     * @param localPart parte local
     */
    public QName(java.lang.String localPart){
        this.namespaceURI = null;
        this.localPart    = localPart;
    }

    /** Constructor con dos parámetros
     * @param namespaceURI espacio de nombres
     * @param localPart parte local
     */
    public QName(java.lang.String namespaceURI,
                 java.lang.String localPart){
        this.namespaceURI = namespaceURI;
        this.localPart    = localPart;
    }

    /** Obtiene el espacio de nombres para esta QName
     * @return Espacio de nombres
     */
    public java.lang.String getNamespaceURI(){
        return this.namespaceURI;
    }

    /** Obtiene la parte local para esta QName
     * @return Parte local
     */
    public java.lang.String getLocalPart(){
        return this.localPart;
    }

    /** Devuelve la representacion String de esta QName
     * @return representación String de esta QName
     */
    public String toString(){
        return ((namespaceURI == null)? localPart
              : '{' + namespaceURI + '}' + localPart);
    }
}
```

```

/** Devuelve como QName la representación de un objeto String
 * @return QName correspondiente al string dado
 * @throws IllegalArgumentException si la cadena especificada
 * no puede convertirse en QName
 */
public static QName valueOf(String s){
    if((s == null)|| (s.equals("")))
        throw new IllegalArgumentException("invalid QName
            literal");

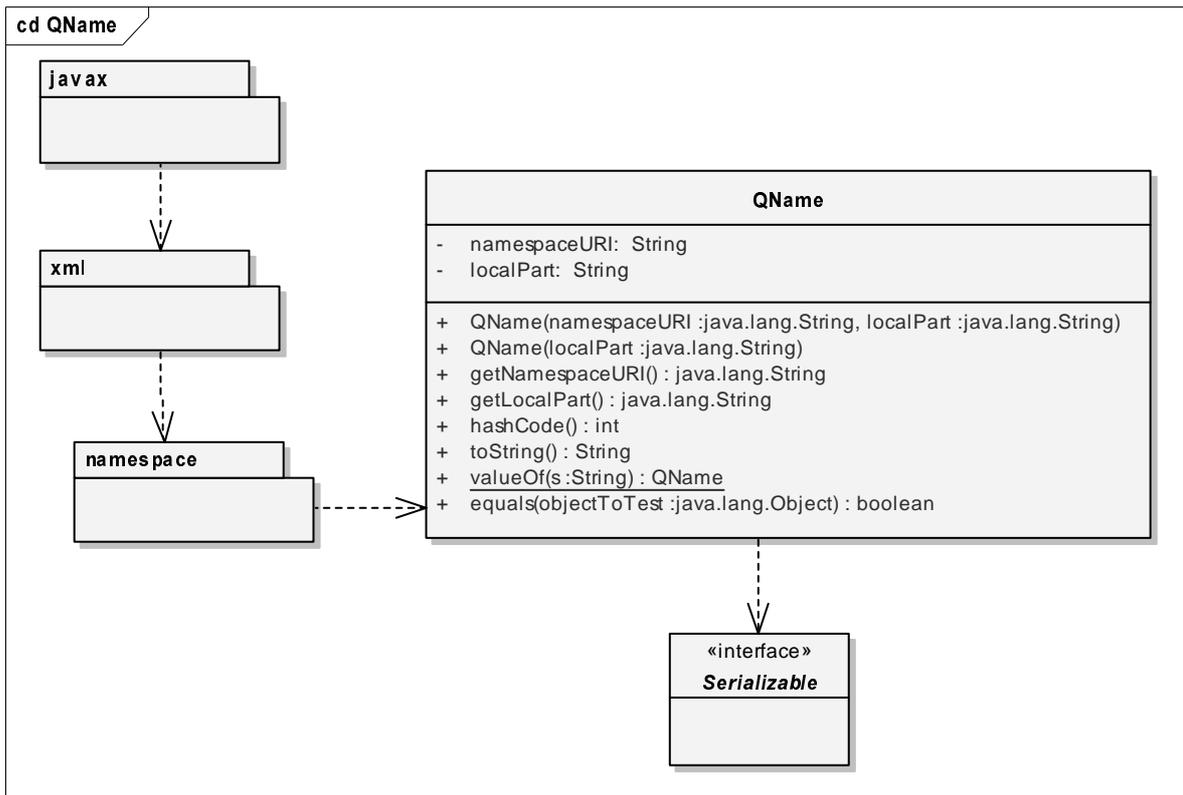
    if(s.charAt(0) == '{'){
        int i = s.indexOf('}');
        if(i == -1){
            throw new IllegalArgumentException("invalid QName
                literal");
        }
        if((i = s.length()) == -1){
            throw new IllegalArgumentException("invalid QName
                literal");
        }
        return new QName(s.substring(1,i),s.substring(i+1));
    }
    else{
        return new QName(s);
    }
}

/** Compara QName con otro objeto
 * @return cierto si el objeto es instancia de QName y si "la parte
 * local" y el "espacio de nombres" coinciden.
 */
public boolean equals(java.lang.Object objectToTest){
    if(objectToTest == this){
        return true;
    }
    if(!(objectToTest instanceof QName)){
        return false;
    }
    if((this.namespaceURI == ((QName)objectToTest).namespaceURI)
    &&(this.localPart == ((QName)objectToTest).localPart)){
        return true;
    }
    return false;
}

/** @return entero que se asocia al objeto QName */
public int hashCode(){
    int valor = 0;
    if(this.namespaceURI != null)
        valor = namespaceURI.hashCode();
    if(this.localPart != null)
        valor += localPart.hashCode();

    return valor;
}
}

```



javax.xml.namespace
Class QName

java.lang.Object

javax.xml.namespace.QName

All Implemented Interfaces:

java.io.Serializable

```

public class QName
extends java.lang.Object
implements java.io.Serializable
  
```

Representa un “nombre cualificado” como es especificado en XML Schema Part2: Datatypes specification. Contiene un namespaceURI, espacio de nombres, y un localPart, parte local.

See Also:

[Serialized Form](#)

Field Summary

private java.lang.String	<u>localPart</u>
private java.lang.String	<u>namespaceURI</u>

Constructor Summary

QName(java.lang.String localPart)
Constructor con un parámetro

QName(java.lang.String namespaceURI, java.lang.String localPart)
Constructor con dos parámetros

Method Summary

boolean	equals (java.lang.Object objectToTest) Compara QName con otro objeto
java.lang.String	getLocalPart () Obtiene la parte local para esta QName
java.lang.String	getNamespaceURI () Obtiene el espacio de nombres para esta QName
int	hashCode ()
java.lang.String	toString () Devuelve la representacion String de esta QName
static QName	valueOf (java.lang.String s) Devuelve como QName la representación de un objeto String

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

QName

public **QName**(java.lang.String localPart)
Constructor con un parámetro

Parameters:

localPart - parte local

QName

public **QName**(java.lang.String namespaceURI,
java.lang.String localPart)
Constructor con dos parámetros

Parameters:

namespaceURI - espacio de nombres
localPart - parte local

Method Detail

getNamespaceURI

public java.lang.String **getNamespaceURI**()
Obtiene el espacio de nombres para esta QName

Returns:

Espacio de nombres

getLocalPart

```
public java.lang.String getLocalPart()
```

Obtiene la parte local para esta QName

Returns:
Parte local

toString

```
public java.lang.String toString()
```

Devuelve la representacion String de esta QName

Returns:
representación String de esta QName

valueOf

```
public static QName valueOf(java.lang.String s)
```

Devuelve como QName la representación de un objeto String

Returns:
QName correspondiente al string dado

Throws:
`java.lang.IllegalArgumentException` - si la cadena especificada no puede convertirse en QName

equals

```
public boolean equals(java.lang.Object objectToTest)
```

Compara QName con otro objeto

Returns:
cierto si el objeto es instancia de QName y si “la parte local” y el “espacio de nombres” coinciden.

hashCode

```
public int hashCode()
```

Returns:
entero que se asocia al objeto QName

Ejercicio 1 (01/09/2001) herencia, excepciones.

El objetivo de este ejercicio es crear nuestra propia excepción mediante la declaración de una clase, de tal forma que se pueda lanzar durante la ejecución del programa para actuar frente a un resultado.

Las excepciones son objetos, por lo tanto se declaran en clases y tienen todas las propiedades de los objetos. La clase de la que heredan todas las excepciones está representado en la caja **A** de la **Figura 1**. Inmediatamente debajo hay dos subclases que dividen las excepciones en dos ramas distintas (caja **B** y **C**).

1. Rellene las cajas A, B y C, de la Figura 1, con el nombre de las clases correspondientes.

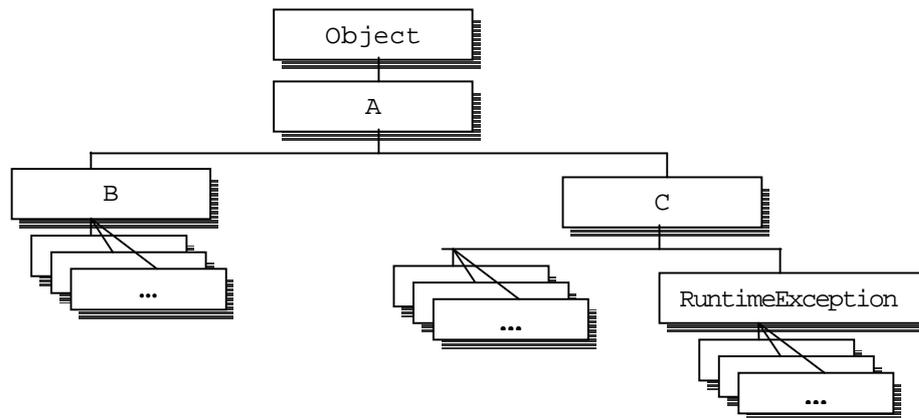


Figura 1. Jerarquía de las Excepciones predefinidas.

2. Para la creación de nuestra excepción asígnele un nombre, especificando si debe heredar o implementar de alguna clase o interfaz.

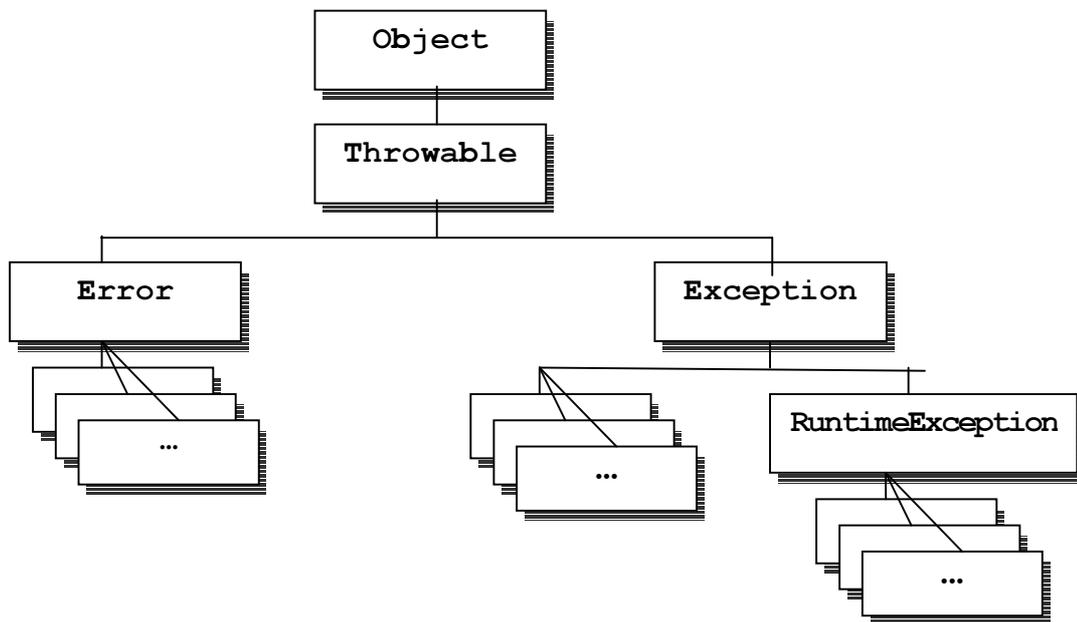
La clase deberá contener una variable instancia privada de nombre **msg**, y de tipo **String**.

3. Implemente el constructor de la clase, al que se le pasa un parámetro de tipo **String**, y que inicializa la variable de instancia.
4. Implemente el método `public String toString()`, de tal forma que devuelva **"Error:"** seguido de la variable de instancia privada.

En la clase `Ejercicio1` se implementan dos métodos uno de ellos es el método `main`, que llamará al método `calcula` que es donde se lanza la excepción si el valor del entero que se le pasa está fuera del rango `[0,10]`, la excepción deberá sacar por pantalla "Fuera de rango", en otro caso se imprimirá por pantalla "Finalización normal". La captura de la excepción se realizará en el método `main` donde se sacará por pantalla el valor del objeto.

5. Implemente el método `static void calcula (int a)`.
6. Implemente el método `main`, con llamadas a la función `calcula` con los valores de 5 y 15.

Solución del Ejercicio1 (apartado 1)



Solución del Ejercicio1 (apartados del 2 al 6)

```
/*
 * Ejercicio1.java
 *
 * Solución al Ejercicio 1 del examen de Septiembre de 2001 de
 * Fundamentos de Telemática.
 */

/**Nuestra excepción. Contiene una variable de instancia
 * de tipo String y privada
 */
class MiExcepcion extends Exception{ //Apartado 2
    private String msg;

    /**Inicializa la variable de instancia.*/
    MiExcepcion(String a){ //Apartado 3
        msg = a;
    }

    /**Devuelve un mensaje que describe la excepción*/
    public String toString(){ //Apartado 4
        return "Error" + msg;
    }
}

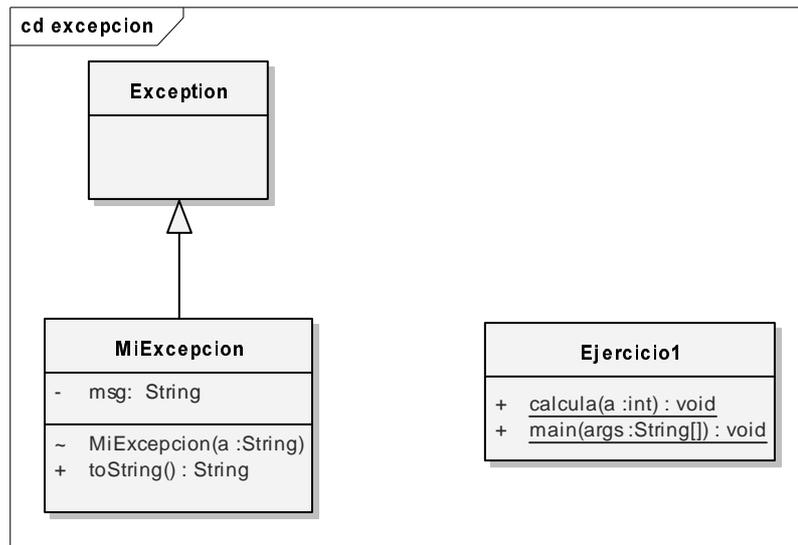
/** La clase Ejercicio1 contiene los métodos main y calcula*/
public class Ejercicio1{
    /** Lanza la excepción si el valor del entero está
     * fuera del rango [0,10]
     */
    public static void calcula (int a) throws MiExcepcion{ //Apartado 5
        if(a>10 || a<0)
            throw new MiExcepcion("Fuera de rango");
        System.out.println("Finalización normal");
    }
}
```

```

/**Llama a la función calcula */
public static void main (String args[]){
    try{
        calcula (5);
        calcula (15);
    }catch(MiExcepcion e){
        System.out.println("Excepción Capturada "+e);
    }
}
}

```

//Apartado 6



Class Ejercicio1

java.lang.Object

Ejercicio1

```

public class Ejercicio1
extends java.lang.Object

```

La clase Ejercicio1 contiene los métodos main y calcula

Constructor Summary

Ejercicio1 ()

Method Summary

static void	calcula (int a) Lanza la excepción si el valor del entero está fuera del rango [0,10]
static void	main (java.lang.String[] args) Llama a la función calcula

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ejercicio1

```
public Ejercicio1()
```

Method Detail

calcula

```
public static void calcula(int a)
    throws MiExcepcion
    Lanza la excepción si el valor del entero está fuera del rango [0,10]
Throws:
    MiExcepcion
```

main

```
public static void main(java.lang.String[] args)
    Llama a la función calcula
```

Ejercicio 2 (01/09/2001) hilos, interfaz, excepciones.

El objetivo de este ejercicio es ver como se puede realizar la comunicación entre dos hilos que se generan en el método principal. Este ejercicio se enmarca en una aproximación al problema clásico de colas, donde un hilo produce datos y otro los consume. Los datos consumidos se marcarán con un entero.

Java proporciona un mecanismo elegante de comunicación entre hilos a través de los métodos `wait()` y `notify()`. Estos métodos pertenecen a la clase `Object`, de manera que todas las clases disponen de ellos.

<code>Final void wait() throws InterruptedException</code>	Le indica al hilo en curso que abandone el monitor y se vaya a dormir hasta que otro hilo entre en el mismo monitor y llame a <code>notify()</code> .
<code>Final void notify() throws IllegalMonitorStateException</code>	Despierta al primer hilo que realizó una llamada a <code>wait()</code> sobre el mismo objeto.

El programa constará de cuatro clases: una para cada hilo (`Productor` y el `Consumidor`), otra clase para la cola (`Q`) y otra en la que estará el método principal (`Ejercicio2`).

La clase `Q` consta de dos métodos sincronizados `synchronized void put (int n)` (que produce los datos) y `synchronized int get ()` (que los consume); y de dos variables de instancia `n` (un entero que representa los datos generados) y `valor`.

La variable de instancia `n` será una secuencia que empiece en 0 y se incremente una unidad cada vez. La variable de instancia `valor` se utilizará para ir alternando la sincronización de ambos métodos, por lo tanto se utilizará para indicar cuando se debe abandonar el monitor. La salida del programa debe ser la siguiente:

```
Produce: 0
Consume: 0
Produce: 1
Consume: 1
...
```

1. Realice la declaración de la clase `Productor` que crea un hilo implementando la interfaz `Runnable`. Esta clase contiene una variable de instancia de tipo `Q` (`Q q;`). El constructor de la clase necesita un parámetro de tipo `Q` para inicializar la variable de instancia. El método `run` es un bucle infinito con la llamada al método `put` de la clase `Q`, y con una variable local de tipo entero que marca los datos producidos.
2. Realice la declaración de la clase `Consumidor` implementando la interfaz `Runnable`. Esta clase contiene una variable de instancia de tipo `Q` (`Q q;`). El constructor de la clase necesita un parámetro de tipo `Q` para inicializar la variable de instancia. El método `run` es un bucle infinito con la llamada al método `get` de la clase `Q`.
3. Implemente la clase `Ejercicio2` que contiene el método `main`. En este método solamente se realiza la instanciación de las 3 clases referidas anteriormente.
4. Implemente el método `put` para que realice lo explicado en este enunciado.
5. Implemente el método `get` para que realice lo explicado en este enunciado.

Solución del Ejercicio2

```
/*
 * Ejercicio2.java
 * Solución al Ejercicio 2 del examen de Septiembre de 2001 de
 * Fundamentos de Telemática.
 */

/** Esta clase contiene 2 métodos sincronizados para
 * que no puedan ejecutarse al mismo tiempo.
 * n representa los datos generados.
 */
class Q{
    int n;
    boolean valor = false;

    /** Consume datos
     * @return entero que marca los datos consumidos
     */
    synchronized int get(){
        if(!valor)
            try{
                wait();
            }catch(InterruptedException e){
                System.out.println("Captura InterruptedException");
            }
        System.out.println("Consume: "+n);
        valor = false;
        notify();
        return n;
    }

    /**Produce datos*/
    synchronized void put(int n){
        if(valor)
            try{
                wait();
            }catch(InterruptedException e){
                System.out.println("Captura InterruptedException");
            }
        this.n = n;
        valor = true;
        System.out.println("Produce: "+n);
        notify();
    }
}

/**Esta clase crea un hilo*/
class Productor implements Runnable{
    Q q;

    /** Inicializa la variable de instancia.*/
    Productor(Q q){
        this.q = q;
        new Thread(this,"Productor").start();
    }

    /**Define la tarea principal del hilo, en este caso, llama a put()*/
    public void run(){
        int i = 0;
        while(i<20)
            q.put(i++);
    }
}
```

```

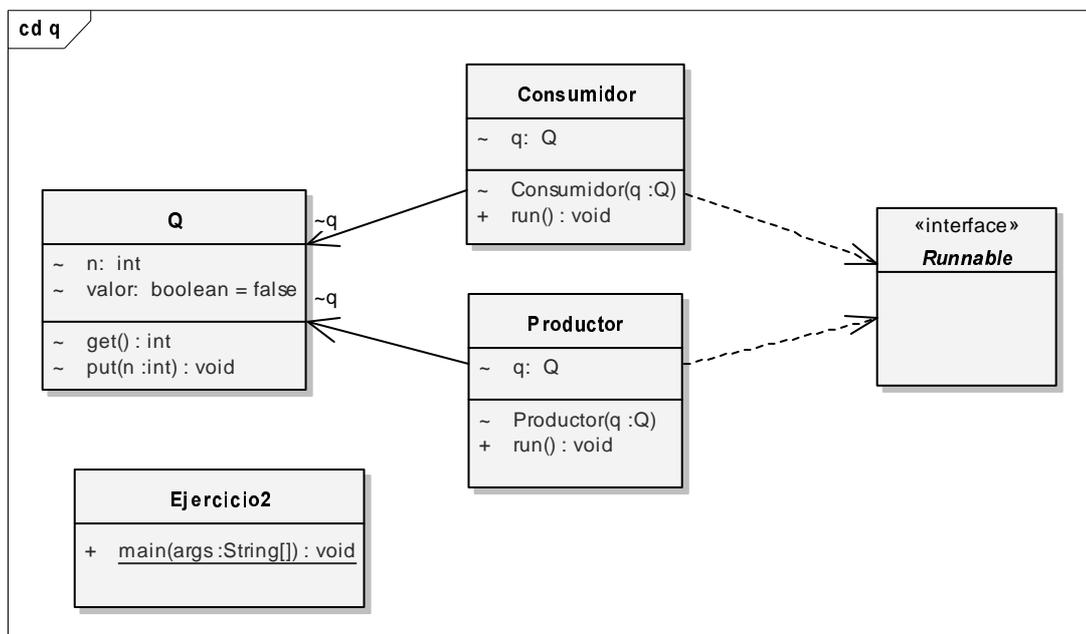
}

/**Esta clase crea un hilo*/
class Consumidor implements Runnable{
    Q q;
    Consumidor(Q q){
        this.q = q;
        new Thread(this,"Consumidor ").start();
    }

    /** Define la tarea principal del hilo, en este caso, llama a get()
     * en un bucle infinito
     */
    public void run(){
        while(true)
            q.get();
    }
}

public class Ejercicio2{
    /**Solo realiza la instanciación de las clases necesarias*/
    public static void main(String args[]){
        Q q =new Q();
        new Productor(q);
        new Consumidor(q);
        System.out.println("Pulse Control-C para finalizar.");
    }
}

```



Class *Ejercicio2*

java.lang.Object
└─ **Ejercicio2**

```
public class Ejercicio2  
extends java.lang.Object
```

Solución al Ejercicio 2 del examen de Septiembre de 2001 de Fundamentos de Telemática.

Constructor Summary

[Ejercicio2\(\)](#)

Method Summary

static void	main (java.lang.String[] args) Solo realiza la instanciación de las clases necesarias
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ejercicio2

```
public Ejercicio2()
```

Method Detail

main

```
public static void main(java.lang.String[] args)  
Solo realiza la instanciación de las clases necesarias
```

Ejercicio 3 (06/09/2001) socket, excepciones.

El objetivo de este ejercicio es escribir un programa que implementa un cliente y un servidor, ejecutándose entre dos puertos diferentes de una máquina local utilizando datagramas. Para utilizar este programa, deberá ejecutar

```
java Ejercicio3
```

de esta forma será el cliente. Después se deberá ejecutar

```
java Ejercicio3 1
```

este será el servidor. Todo lo que se introduzca en la ventana del servidor se enviará a la ventana del cliente después de recibir un carácter de nueva línea. El programa utilizará dos clases **DatagramSocket** y **DatagramPacket**. Este ejercicio constará de una sola clase.

1. ¿Cuál es el rango para los puertos permitido para realizar la conexión usando datagramas? Implemente los números de los puertos para el cliente y para el servidor, la declaración de una variable **DatagramSocket**, y una matriz de **byte** de tamaño 1024 como variables de instancia estáticas.
2. ¿Qué paquete (o paquetes) necesita importar?
3. Implemente el método principal **main** que realice la llamada a los métodos **ElCliente** y **ElServidor** de acuerdo con la llamada al intérprete que aparece arriba. El método principal deberá capturar las excepciones que sean necesarias e instanciar los objetos de la clase **DatagramSocket** utilizando la siguiente tabla.

<i>Constructores de la clase DatagramSocket</i>
DatagramSocket() throws SocketException
DatagramSocket(int puerto) throws SocketException
DatagramSocket(int Puerto, InetAddress laddr) throws SocketException

<i>Constructores de la clase DatagramPacket</i>
DatagramPacket(byte[] buf, int length)
DatagramPacket(byte[] buf, int length, InetAddress address, int Puerto)
DatagramPacket(byte[] buf, int offset, int length,)
DatagramPacket(byte[] buf, int offset, int length, InetAddress addr, int puerto)

El método estático **InetAddress.getLocalHost()** devuelve la dirección de la máquina local.

Los métodos de la clase **DatagramSocket** para enviar y recibir los datagramas son:

void send (DatagramPacket p) Envía un datagrama a través de un socket.

void receive (DatagramPacket p) Recibe un datagrama de un socket.

Pueden ser útiles los siguientes métodos de la clase **DatagramPacket**, y los constructores de **String**:

Byte [] getData () Devuelve los datos recibidos o los datos enviados.

Int getLength () Devuelve la longitud de los datos (recibidos o enviados).

String(byte[] bytes, int offset, int length) Constructor.

String(byte[] ascii, int hibyte, int offset, int count) Constructor.

4. Implemente el método **public static void ElServidor() throws Exception**. En este método se lee de la entrada estándar carácter a carácter, se coloca en la tabla de **byte**, se construye el paquete con el constructor apropiado y se envía con el método **send**. Si el carácter leído de la entrada estándar es el **'\n'** se ha acabado de llenar el paquete y se envía. Finaliza el método cuando se lee e valor **-1**.
5. Implemente el método **public static void ElCliente() throws Exception** en el que en un bucle infinito se construya un objeto **DatagramPacket** y se reciba la información y se imprima por pantalla.

Solución del Ejercicio3

1. El rango de los números de puerto va desde 0 hasta 65 535 ya que los puertos se representan por números de 16 bits. Los puertos en el rango de números de 0-1023 está restringido; están reservados para uso de los servicios tales como http, ftp y otros servicios del sistema. Las aplicaciones no deben ligarse a esos puertos.

```
/*
 * Ejercicio3.java
 *
 * Solución al Ejercicio 3 del examen de Septiembre de 2001 de
 * Fundamentos de Telemática.
 */

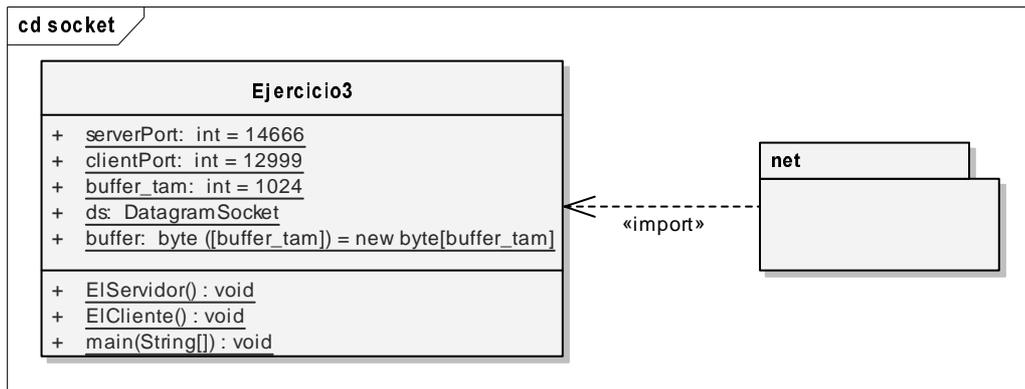
import java.net.*;                                     //Apartado 2

public class Ejercicio3{
    public static int serverPort = 14666;
    public static int clientPort = 12999;
    public static int buffer_tam = 1024;
    public static DatagramSocket ds;
    public static byte buffer[] = new byte[buffer_tam];

    public static void ElServidor() throws Exception{    //Apartado 4
        int pos = 0, c;
        do{
            c = System.in.read();
            switch(c){
                case '\n':
                    ds.send(new DatagramPacket(buffer,pos,
                                                InetAddress.getLocalHost(),clientPort));
                    pos = 0;
                    break;
                default:
                    buffer[pos++] = (byte)c;
            }
        }while(c!=-1);
    }

    public static void ElCliente() throws Exception {    //Apartado 5
        while(true){
            DatagramPacket p = new DatagramPacket (buffer,
                                                    buffer.length);
            ds.receive(p);
            System.out.println(new
                String(p.getData(),0,0,p.getLength()));
        }
    }

    public static void main(String args[]) throws Exception{ //Apartado 3
        if(args.length == 1){
            ds = new DatagramSocket(serverPort);
            ElServidor();
        }else{
            ds = new DatagramSocket(clientPort);
            ElCliente();
        }
    }
}
```



Package **Class** **Tree** **Deprecated** **Index** **Help**

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Class Ejercicio3

java.lang.Object

└─ Ejercicio3

```
public class Ejercicio3
extends java.lang.Object
```

Solución al Ejercicio 3 del examen de Septiembre de 2001 de Fundamentos de Telemática.

Field Summary

static byte[]	<u>buffer</u>
static int	<u>buffer tam</u>
static int	<u>clientPort</u>
static java.net.DatagramSocket	<u>ds</u>
static int	<u>serverPort</u>

Constructor Summary

Ejercicio3()

Method Summary

static void	<u>ElCliente()</u>
static void	<u>ElServidor()</u>
static void	<u>main</u> (java.lang.String[] args)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

serverPort

public static int **serverPort**

clientPort

public static int **clientPort**

buffer_tam

public static int **buffer_tam**

ds

public static java.net.DatagramSocket **ds**

buffer

public static byte[] **buffer**

Constructor Detail

Ejercicio3

public **Ejercicio3**()

Method Detail

ElServidor

public static void **ElServidor**()
throws java.lang.Exception

Throws:
java.lang.Exception

ElCliente

public static void **ElCliente**()
throws java.lang.Exception

Throws:
java.lang.Exception

main

public static void **main**(java.lang.String[] args)
throws java.lang.Exception

Throws:
java.lang.Exception

Ejercicio 1 (6/02/2002) E/S

El objetivo de este ejercicio es escribir un programa en Java que convierta caracteres en mayúsculas usando un filtro (**Filter**) de entrada. Los caracteres se suministran como el primero de los argumentos en la línea de comando y después se realiza el filtrado sobrescribiendo los métodos de la clase **Filter** (el método estático **toUpperCase** de la clase **Character**, pasa un carácter a mayúsculas. La clase **Character** es la clase que “envuelve” al tipo de dato simple **char**).

Los flujos **Filter** hacen posible encadenar flujos produciendo compuestos de superior utilidad. Todo flujo de filtro (**Filter**) está ligado a otro flujo en el que delega las acciones de entrada y/o de salida reales. En este ejercicio la acción de entrada está ligada a **StringReader** para leer los caracteres de la línea de comandos.

1. Los flujos **Filter** pueden ser de bytes o de caracteres. ¿Qué flujos son más apropiados utilizar en este caso, los bytes (**FileInputStream** y **FileOutputStream**) o los de caracteres (**FilterReader** y **FilterWriter**)? ¿Por qué? (0,25p).
2. Paquete(s) que se necesita(n) importar (0,25p).

Se pretende construir una clase (**ConversorMayusculas**) que herede los métodos de **FilterReader** y que redefina (sobrescriba) los dos métodos **read** de la clase **FilterReader** y contenga un constructor que sea invocado con un parámetro de tipo **Reader**. En los siguientes puntos se realiza la declaración completa de la clase **ConversorMayusculas**.

1. Declare el nombre y el constructor de la clase usando la invocación al constructor de la superclase (1,5p).
2. Sobrescriba el método **read** (`int read() throws IOException`) de la clase **FilterReader**, para que, mediante el uso del método de la superclase, se realice la operación de filtrado (pasar a mayúsculas una letra si no lo es) (3p).
3. Sobrescriba el método **read** (`int read(char[] cbuf, int off, int len) throws IOException`) de la clase **FilterReader**, para que, mediante el uso del método de la superclase, se realice la operación de filtrado (pasar a mayúsculas una letra si no lo es) (3p).

Como fuente de datos se utilizará una cadena de texto que se proporciona como el primer argumento en la línea de comandos para crear un objeto **StringReader**. Este objeto se utilizará como parámetro para el constructor del objeto que realiza el filtrado. Todo esto se debe realizar en una clase de nombre **Ejercicio1** que implementará el método principal.

4. Declare una clase que contenga el método principal (**main**), en el que se realice la instanciación de los objetos necesarios para realizar el filtrado proporcionando los caracteres desde la línea de comandos y se saque la información por pantalla (2p). Por ejemplo:

```
C:\jdk1.2.2\bin>java Ejercicio1 Sevilla
SEVILLA
C:\jdk1.2.2\bin>
```

Clase: Character

`static char toUpperCase(char ch)`. Devuelve el carácter en mayúsculas del argumento si lo tuviera, sino devuelve el propio carácter.

Clase: FilterReader

Constructor: <code>FilterReader(Reader in)</code>

<code>void close()</code> throws <code>IOException</code>

<code>void mark(int readAheadLimit)</code> throws <code>IOException</code>
--

<code>boolean markSupported()</code> throws <code>IOException</code>
--

<code>int read()</code> thors <code>IOException</code>
--

Este método lee un carácter como un entero, devuelve -1 si se alcanza el final del flujo.

<code>int read(char[] cbuf, int off, int len)</code> throws <code>IOException</code>
--

Devuelve el número de caracteres leídos ó -1 si se alcanza el final del flujo.
--

<code>boolean ready()</code> throws <code>IOException</code>
--

<code>void reset()</code> throws <code>IOException</code>

<code>long skip(long n)</code>

Clase: StringReader

Constructor: <code>StringReader(String s)</code>
--

Solución del Ejercicio1

```
/*
 * Ejercicio1.java
 *
 * Solución al ejercicio 1 del examen de Febrero de 2002 de
 * Fundamentos de Telemática.
 */

import java.io.*;

/**
 * La clase ConversorMayusculas hereda de la clase FilterReader
 */
class ConversorMayusculas extends FilterReader{
    ConversorMayusculas (Reader in){
        super(in);
    }
}

/**
 * pone en mayúsculas siempre que no haya alcanzado el final del argumento
 */
public int read() throws IOException{
    int c = super.read();
    return (c!=-1 ? c : Character.toUpperCase((char)c));
}

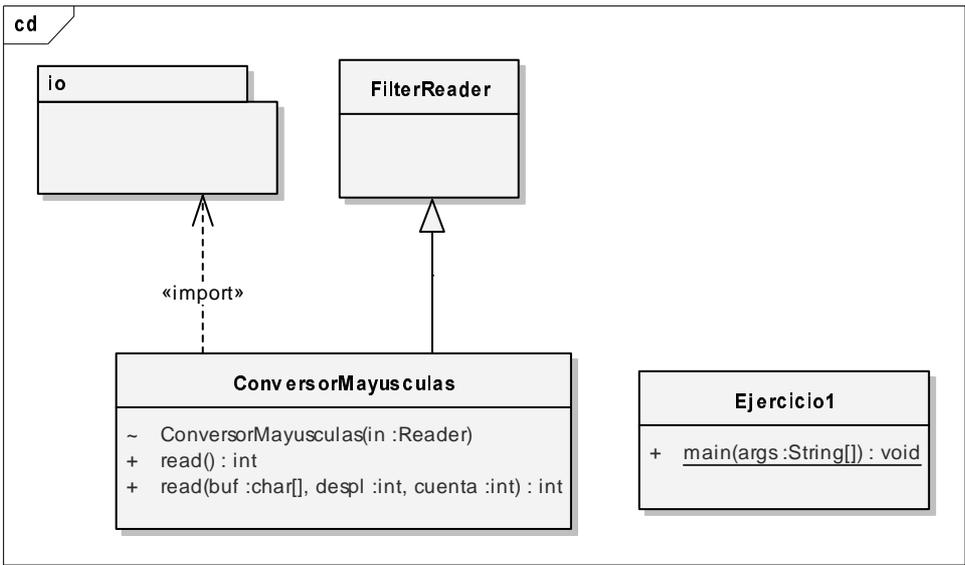
/**
 * pone en mayúsculas siempre que no haya alcanzado el final del argumento
 */
public int read(char[] buf, int displ, int cuenta) throws IOException{
    int nread = super.read(buf,displ,cuenta);
    int ult = displ + nread;
    for(int i=displ; i<ult;i++)
        buf[i] = Character.toUpperCase(buf[i]);
    return nread;
}
}

/**
 * Clase Ejercicio1 que contiene el método Main
 */
public class Ejercicio1 {

    /**
     * @param args Argumentos de la línea de comandos
     */
    public static void main(String args[])throws IOException{
        StringReader fuen = new StringReader(args[0]);

        FilterReader f    = new ConversorMayusculas(fuen);

        int c;
        while((c = f.read()) != -1)
            System.out.print((char)c);
        System.out.println();
    }
}
}
```



Class Ejercicio1

java.lang.Object
└─Ejercicio1

```
public class Ejercicio1
    extends java.lang.Object
```

Solución al ejercicio 1 del examen de Febrero de 2002 de Fundamentos de Telemática.

Constructor Summary

Ejercicio1()	
------------------------------	--

Method Summary

static void	main (java.lang.String[] args)
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ejercicio1

```
public Ejercicio1()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
    throws java.io.IOException
```

Parameters:

args - Argumentos de la línea de comandos

Throws:

java.io.IOException

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

Ejercicio 2 (6/02/2002) hilos, excepciones

El objetivo de este ejercicio es escribir un programa en Java que utilice la sincronización de hilos (entre un hilo productor y otro consumidor, llamados desde el hilo principal) como aproximación al problema de colas. Los datos producidos/consumidos se marcarán con un entero; el hilo productor deposita una secuencia de números (0, 1, 2,...) y el hilo consumidor lee estos datos. Se imprimirá por pantalla tanto cuando se producen como cuando se consumen. Una mejora considerable supone utilizar una *cola circular* de tal forma que se puedan “producir” varios datos antes de tener que “consumir”.

El programa constará de cuatro clases: una para cada hilo (**Productor/Consumidor**), otra para la cola (**Q**) y otra en la que estará el método principal (**Ejercicio2**).

La clase **Q** conste de dos métodos sincronizados **public synchronized void put(int val)** (que produce los datos) y **public synchronized int get()** (que los consume); y de las siguientes variables de instancia: **buffer** que es una matriz de enteros de cinco elementos que se usa como *cola circular*¹ para colocar los datos producidos, **leible** indica si un consumidor puede o no leer de la *cola circular*, **escribible** indica si un productor puede o no escribir en la *cola circular*; **posicionLectura** indica la posición de la que el consumidor puede leer el siguiente valor, **posicionEscritura** indica la siguiente posición en la que un productor puede colocar un valor. Si no se puede realizar la operación de lectura o escritura en la *cola circular* se esperará el hilo (**wait()**) hasta que se pueda.

El mecanismo de comunicación entre hilos se realizará a través de los métodos **wait()**, **notify()** y **sleep(long millis)**.

1. Declare la clase **Productor** que hereda de la clase **Thread**. Esta clase contiene una variable de instancia de tipo **Q (Q q;)**. El constructor de la clase necesita un parámetro de tipo **Q** para inicializar la variable de instancia. El método **run** es un bucle infinito con la llamada al método **put** de la clase **Q**, y con una variable local de tipo entero que marca los datos producidos. Una vez incrementado e entero se duerme (**sleep()**) el hilo un instante aleatorio(**random()**) de valor máximo 100 (1,5p).
2. Declare la clase **Consumidor** que hereda de la clase **Thread**. Esta clase contiene una variable de instancia de tipo **Q (Q q;)**. El constructor de la clase necesita un parámetro de tipo **Q** para inicializar la variable de instancia. El método **run** es un bucle infinito con la llamada al método **get** de la clase **Q**. Una vez consumido el entero se duerme el hilo un instante aleatorio de valor máximo 100 (1,5p).
3. Implemente la clase **Ejercicio2** que contiene el método **main**. En él se realizará la instanciación de las otras clases (1,5p).
4. Implemente la clase **Q** con los métodos **put** y **get** para que realice lo explicado en este enunciado (3p).
5. ¿Qué sucede si los métodos **put** y **get** no estuvieran sincronizados) (1,5p).
6. ¿Se produce o se puede producir bloqueo en algún momento? (1p).

¹ Las **colas** son estructuras que tienen una disciplina de extracción/inserción *FIFO (First In, First Out)*, donde el elemento más antiguo almacenado es el primero en ser recuperado. En las **colas circulares** se reutilizan las posiciones del vector (**buffer** en el ejercicio) que se van extrayendo. Si se llega al final del vector se continuará de nuevo con el primer elemento del vector. En las colas circulares la condición de *cola vacía* y de *cola llena* es la misma (para nuestro ejercicio **posicionLectura = posicionEscritura**, por lo tanto se utilizará **leible/escribible** indicando si se puede leer/escribir de/en la *cola circular*).

Clase: Object

<code>final void wait () throws InterruptedException</code>	Le indica al hilo en curso que abandone el monitor y se vaya a dormir hasta que otro hilo entre en el mismo monitor y llame a <code>notify()</code> .
<code>final void notify () throws InterruptedException</code>	Despierta al primer hilo que realizó una llamada a <code>wait()</code> sobre el mismo objeto.

Clase: Thread

<code>static void sleep (long millis) throws InterruptedException</code>	Duerme temporalmente un hilo los milisegundos que se especifican.
<code>static void sleep (long millis, int nanos) throws InterruptedException</code>	Duerme temporalmente un hilo los milisegundos más los nanosegundos que se especifican.

Clase: Math (del paquete java.lang)

<code>static double random () throws InterruptedException</code>	Devuelve un valor double como un positivo mayor o igual que 0 y menor o igual que 1
--	---

Solución del Ejercicio2

```
/*
 * Ejercicio2.java
 *
 * Solución al Ejercicio 2 del examen de Febrero de 2002 de
 * Fundamentos de Telemática.
 */

/** Clase Q correspondiente a la cola circular */
class Q {
    private int buffer[] = {9,9,9,9,9};
    private boolean leible = false;
    private boolean escribible = true;
    private int posicionLectura = 0;
    private int posicionEscritura = 0;
    String output = null;

    public Q (String out){
        output = out;
    }

    /** Método put que produce los datos */
    public synchronized void put(int val){
        while(!escribible){
            try{
                output += ("ESPERANDO PARA PRODUCIR "+val);
                wait();
            }catch(InterruptedException e){
                System.err.println("Excepción: " + e);
            }
        }

        buffer[posicionEscritura] = val;
        leible = true;
        output += ("\nProduje "+ val +" en la celda "+ posicionEscritura);
        posicionEscritura = ++posicionEscritura % 5;
        output += ("\twrite" + posicionEscritura + "\thread" +
            posicionLectura);

        printBuffer(output, buffer);
        if(posicionEscritura == posicionLectura){
            escribible = false;
            output += ("\nBUFFER LLENO");
        }
        notify();
    }

    /** Método get que consume los datos */
    public synchronized int get(){
        int val;

        while(!leible){
            try{
                output += ("ESPERANDO PARA CONSUMIR");
                wait();
            }catch(InterruptedException e){
                System.err.println("Excepción: " + e);
            }
        }
    }
}
```

```

        escribible = true;
        val = buffer[posicionLectura];
        output +=("\nConsumí " + val + " de la celda " + posicionLectura );
        posicionLectura = ++posicionLectura % 5;
        output += ("\twrite " + posicionEscritura + "\tread" +
                posicionLectura );
        printBuffer(output, buffer);
        if(posicionEscritura == posicionLectura){
            leible = false;
            output += ("\nBUFFER VACÍO");
        }
        notify();
        return val;
    }

    /** Método printBuffer que saca por pantalla el buffer actual */
    public void printBuffer( String output, int buf[]){
        System.out.print(output + "\tbuffer: ");

        for(int i = 0; i<buf.length; i++)
            System.out.println(" "+buf[i]);
    }
}

/** Clase Productor hereda de Thread */
class Productor extends Thread {
    private Q q;

    public Productor (Q q) {
        this.q = q;
    }

    /** invoca un número de veces al método put de la clase Q */
    public void run() {
        for(int count = 0; count<10; count++){
            q.put(count);

            try{
                sleep((int)(Math.random()*100));
            }catch(InterruptedException e){
                System.err.println("Excepción " + e.toString());
            }
        }
    }
}

/** Clase Consumidor hereda de Thread */
class Consumidor extends Thread {
    private Q q;

    public Consumidor (Q q) {
        this.q = q;
    }

    /** invoca un número de veces al método get de la clase Q */
    public void run() {
        int val;

        val = q.get();
        while(val != 9){

```

```

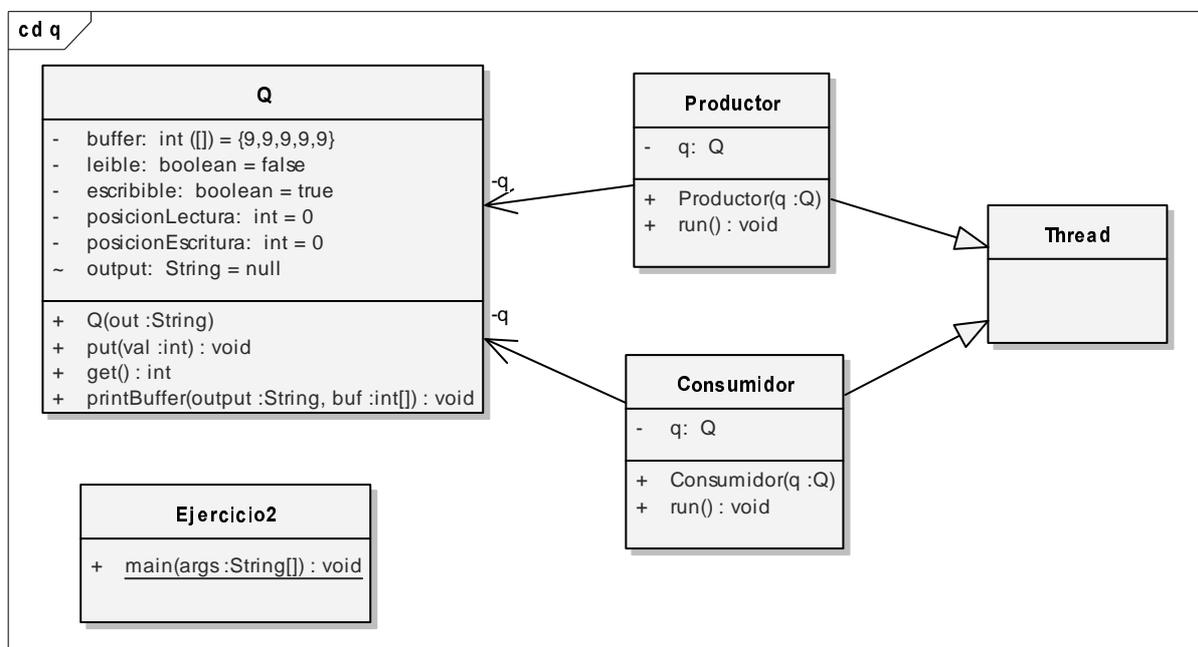
        try{
            sleep((int) (Math.random()*100));
        }catch(InterruptedException e){
            System.err.println("Excepción " + e.toString());
        }
        val = q.get();
    }
}

/**
 * Clase Ejercicio2 que contiene al método main.
 * Comienza el trasvase de información entre consumidor y productor.
 */
public class Ejercicio2 {

    /**
     * @param args Argumentos de la línea de comandos
     */
    public static void main(String args[]){
        String output = null;
        Q h = new Q (output);
        Productor p = new Productor(h);
        Consumidor c = new Consumidor(h);

        p.start();
        c.start();
    }
}

```



Class Ejercicio2

java.lang.Object
└─ Ejercicio2

```
public class Ejercicio2
    extends java.lang.Object
```

Solución al ejercicio 2 del examen de Febrero de 2002 de Fundamentos de Telemática.

Constructor Summary

Ejercicio2()	
------------------------------	--

Method Summary

static void	main (java.lang.String[] args)
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ejercicio2

```
public Ejercicio2()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Parameters:

args - Argumentos de la línea de comandos

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

Ejercicio 3 (6/02/2002) paquetes, intérprete

Este ejercicio se contesta parte sobre el enunciado y parte sobre una hoja aparte. Se proporciona el siguiente código, contenido en los ficheros que se indican como comentario:

<pre>package p1; //Fichero: Proteccion.java public class Proteccion { int n = 1; private int n_pri = 2; protected int n_pro = 3; public int n_pub = 4; public Proteccion() { System.out.println("constructor base "); System.out.println("n = " + n); System.out.println("n_pri = " + n_pri); System.out.println("n_pro = " + n_pro); System.out.println("n_pub = " + n_pub); } }</pre>	<pre>//Fichero: Derivada.java class Derivada extends Proteccion { Derivada() { System.out.println("construct. Deriv "); System.out.println("n = " + n); System.out.println("n_pri = " + n_pri); System.out.println("n_pro = " + n_pro); System.out.println("n_pub = " + n_pub); } }</pre>
<pre>//Fichero: MismoPaquete.java class MismoPaquete { MismoPaquete () { Proteccion p = new Proteccion(); System.out.println("construct. MismoPaquete "); System.out.println("n = " + p.n); System.out.println("n_pri = " + p.n_pri); System.out.println("n_pro = " + p.n_pro); System.out.println("n_pub = " + p.n_pub); } }</pre>	

1. ¿Qué errores se presentan en el código si se desea que los anteriores ficheros pertenezcan al mismo paquete? (*Conteste sobre la hoja de enunciado. Ponga la línea como comentario si sobra y añada el código si falta*).
2. Especifique cuáles son las órdenes de compilación del JDK de los ficheros que se proporcionan. Indique todos los detalles.
3. Escriba el código Java de una clase con el método `main` que instancie todas las clases anteriores.
4. Indique cuál es la llamada al intérprete y el resultado del código escrito en el apartado 3.

<pre>package p2; //Fichero: Proteccion0.java class Proteccion2 extends Proteccion { Proteccion2() { System.out.println("construct Proteccion2 "); System.out.println("n = " + n); System.out.println("n_pri = " + n_pri); System.out.println("n_pro = " + n_pro); System.out.println("n_pub = " + n_pub); } }</pre>	<pre>//Fichero: Proteccion0 continuacion ... class OtroPaquete { OtroPaquete () { Proteccion p = new Proteccion(); System.out.println("construct. OtroPaquete "); System.out.println("n = " + p.n); System.out.println("n_pri = " + p.n_pri); System.out.println("n_pro = " + p.n_pro); System.out.println("n_pub = " + p.n_pub); } }</pre>
---	---

5. ¿Qué errores se presentan en el código si se desea que los anteriores ficheros pertenezcan a otro paquete? (*Se contesta sobre la hoja de enunciado*).
6. Especifique cuáles son las órdenes de compilación de los ficheros que se proporcionan. Indique todos los detalles.
7. Escriba el código Java de una clase con el método `main` que instancie todas las clases anteriores.
8. Indique cuál es la llamada al intérprete y el resultado del código escrito en el apartado 7.

Solución del Ejercicio3

```
package p1;
//Fichero: Proteccion.java
public class Proteccion {
    int n = 1;
    private    int n_pri = 2;
    protected int n_pro = 3;
    public    int n_pub = 4;

    public Proteccion() {
        System.out.println("constructor base ");
        System.out.println("n = " + n);
        System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

Orden de compilación de Proteccion.java:

C:\jdk1.2.2\bin>javac .p1\Proteccion.java

```
package p1;
//Fichero: Derivada.java

class Derivada extends Proteccion {
    Derivada() {
        System.out.println("construct. Deriv ");
        System.out.println("n = " + n);

        //sólo para su clase
        //System.out.println("n_pri = " + n_pri);

        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

Orden de compilación de Derivada.java:

C:\jdk1.2.2\bin>javac .p1\Derivada.java

```
package p1;
//Fichero: MismoPaquete.java

class MismoPaquete {
    MismoPaquete () {
        Proteccion p = new Proteccion();
        System.out.println("construct. MismoPaquete ");
        System.out.println("n = " + p.n);

        //sólo para su clase
        //System.out.println("n_pri = " + p.n_pri);

        System.out.println("n_pro = " + p.n_pro);
        System.out.println("n_pub = " + p.n_pub);
    }
}
```

Orden de compilación de MismoPaquete.java:

C:\jdk1.2.2\bin>javac .p1\ MismoPaquete.java

```
package p1;
//Fichero Demo.java

public class Demo {
    public static void main(String args[]){
        Proteccion ob1 = new Proteccion();
        Derivada ob2 = new Derivada();
        MismoPaquete ob3 = new MismoPaquete();
    }
}
```

Orden de compilación de Demo.java:

```
C:\jdk1.2.2\bin>javac .p1\ Demo.java
```

Llamada al intérprete y resultado:

```
C:\jdk1.2.2\bin>java p1.Demo
```

```
constructor base
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
constructor base
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
construct. Deriv
n = 1
n_pro = 3
n_pub = 4
constructor base
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
construct. MismoPaquete
n = 1
n_pro = 3
n_pub = 4
```

```

package p2;
import pl.*;
//Fichero: Proteccion0.java

class Proteccion2 extends Proteccion {
    Proteccion2() {
        System.out.println("constructor Proteccion2 ");

        //solo para su clase o paquete
        //System.out.println("n = " + n);

        //solo para su clase
        //System.out.println("n_pri = " + n_pri);

        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}

class OtroPaquete {
    OtroPaquete() {
        Proteccion p = new Proteccion();
        System.out.println("construct. OtroPaquete ");

        //solo para su clase o paquete
        //System.out.println("n = " + p.n);

        //solo para su clase
        //System.out.println("n_pri = " + p.n_pri);

        //solo para su clase, subclase o paquete
        //System.out.println("n_pro = " + p.n_pro);

        System.out.println("n_pub = " + p.n_pub);
    }
}

```

C:\jdk1.2.2\bin>javac .p2\Proteccion0.java

```

package p2;
//Fichero: Ppal.java

class Ppal {
    public static void main (String args[]){
        Proteccion2 ob1 = new Proteccion2();
        OtroPaquete ob3 = new OtroPaquete();
    }
}

```

C:\jdk1.2.2\bin>javac .p2\Ppal.java

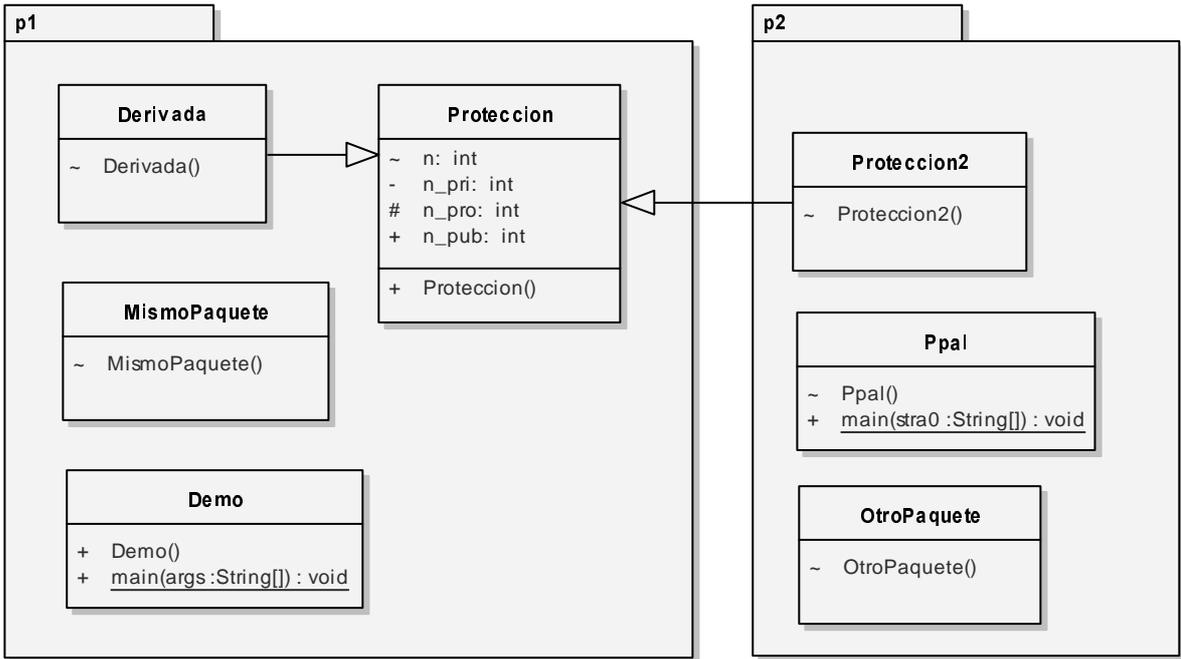
La llamada al intérprete:

```

C:\jdk1.2.2\bin>java p2.Ppal
constructor base
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
constructor Proteccion2
n_pro = 3
n_pub = 4
constructor base
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
construct. OtroPaquete
n_pub = 4

```

cd pro



Package *p1*

Class Summary

Demo	
Derivada	
MismoPaquete	
Proteccion	

p1

Class **Demo**

java.lang.Object

p1.Demo

```
public class Demo
extends java.lang.Object
```

Constructor Summary

Demo()	
------------------------	--

Method Summary

static void	main (java.lang.String[] args)
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Demo

```
public Demo()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

p1

Class Derivada

java.lang.Object

p1.Proteccion

p1.Derivada

class **Derivada**
extends Proteccion

Field Summary

Fields inherited from class p1.Proteccion

n, n_pro, n_pub

Constructor Summary

(package private)	<u>Derivada</u> ()
-------------------	--------------------

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Derivada

Derivada()

p1

Class MismoPaquete

java.lang.Object

p1.MismoPaquete

class **MismoPaquete**
extends java.lang.Object

Constructor Summary

(package private)	<u>MismoPaquete</u> ()
-------------------	------------------------

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MismoPaquete

MismoPaquete()

p1

Class Proteccion

java.lang.Object

p1.[Proteccion](#)

Direct Known Subclasses:

[Derivada](#)

public class **Proteccion**

extends java.lang.Object

Field Summary

(package private) int	<u>n</u>
private int	<u>n_pri</u>
protected int	<u>n_pro</u>
int	<u>n_pub</u>

Constructor Summary

Proteccion ()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

n

int n

n_pri

private int n_pri

n_pro

protected int n_pro

n_pub

public int n_pub

Constructor Detail

Proteccion

public Proteccion()

[Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

Package p2

Class Summary

OtroPaquete	
Ppal	
Proteccion2	

p2

Class OtroPaquete

java.lang.Object

p2.OtroPaquete

class **OtroPaquete**

extends java.lang.Object

Constructor Summary

(package private)	OtroPaquete ()
-------------------	--------------------------------

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

OtroPaquete

OtroPaquete()

p2

Class Ppal

java.lang.Object

p2.Ppal

class Ppal

extends java.lang.Object

Constructor Summary

(package private)	Ppal()
-------------------	------------------------

Method Summary

static void	main (java.lang.String[] args)
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ppal

Ppal()

Method Detail

main

public static void **main**(java.lang.String[] args)

p2

Class Proteccion2

java.lang.Object

Proteccion

class **Proteccion2**
extends Proteccion

Constructor Summary

(package private)	Proteccion2 ()
-------------------	--------------------------------

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Proteccion2

Proteccion2()

[Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Ejercicio 1 (21/06/2002) E/S, hilos, excepciones.

El objetivo de este ejercicio es escribir un programa en Java que utilice flujos **Piped** para conectar un hilo que genere texto con otro hilo que lea el texto generado. Al igual que gran parte de los flujos en Java, los flujos **Piped** se pueden clasificar en flujos de *bytes* (**PipedInputStream** y **PipedOutputStream**) y en flujos de *caracteres* (**PipedReader** y **PipedWriter**)

Los flujos **Piped** se utilizan como parejas de entrada / salida. Los datos que se escriben en el flujo de salida de una pareja son los datos que se leen del flujo de entrada. El conducto de transmisión mantiene un *buffer* interno (*no se puede controlar el tamaño*), con una capacidad definida por la implementación, que permite que la escritura y la lectura se realicen a diferentes velocidades.

La pareja de entrada / salida debe estar unida entre sí. Se pueden unir de dos formas:

- ✓ Haciendo que un objeto **PipedWriter** sea el parámetro para el constructor del **PipedReader** (o viceversa).
- ✓ Con el método **connect**, El método **connect** de **PipedReader** toma un parámetro de **PipedWriter**, y viceversa. No importa si se conecta “y con x” o “x con y”, y el resultado es el mismo.

La única forma segura de utilizar flujos **Piped** es mediante dos hilos: uno para leer y otro para escribir. Al escribir un extremo se bloquea el hilo cuando “se llena el conducto de transmisión”. Al leer de un “conducto de transmisión” se bloquea el hilo hasta que no haya entrada disponible. El “conducto” lleva cuenta de las actualizaciones de los hilos lector y escritor, para evitar que un hilo se bloquee para siempre; y comprueba si el hilo del otro extremo está vivo antes de bloquear el hilo actual. Si el hilo del otro extremo ha terminado, el hilo actual recibirá una **IOException**.

El programa consta de tres clases: **GeneradorTexto** que implemente el hilo que genera el texto, **HiloLector** que implemente un hilo que lee el texto generado y **Ejercicio1** que instancia los objetos para que se pueda ejecutar correctamente.

Se pide:

- 1.- Paquetes que se necesita importar.
- 2.- Implementar la clase **GeneradorTexto**, con el constructor apropiado, y con el método **run** (**public void run**), que genera, mediante un hilo, todo el abecedario.
- 3.- ¿Qué líneas de código hay que sustituir en el apartado anterior para que lea los caracteres de la entrada estándar?
- 4.- Implementar la clase **HiloLector** con el constructor apropiado y del método **run** (**public void run**), que lea el texto generado y lo escriba en la salida.
- 5.- Declarar la clase **Ejercicio1** que instancie los objetos necesarios para que realice el algoritmo explicado en este ejercicio. Tenga en cuenta la finalización de los hilos.
- 6.- En el caso anterior hay tres hilos (el que escribe, el que lee y el del método **main**). Rescribir la clase **Ejercicio1** para que instancie solo a **GeneradorTexto** y la lectura se haga en el hilo principal.
- 7.- Si el hilo lector y el escritor son el mismo se produce alguna situación no deseable. Explíquela.

Clase: PipedWriter

Constructor:

```
public PipedWriter(PipedReader s) throws IOException
```

Crea un flujo escritor conectado a uno lector.

```
public PipedWriter()
```

Métodos:

```
void close() throws IOException
```

```
void connect(PipedReader s) throws IOException
```

Conecta un conducto escritor a uno lector. Si el objeto ya está conectado a algún otro lector, lanza la excepción.

```
void flush() throws IOException
```

```
void write(char[] cbuf, int off, int len) throws IOException
```

```
void write(int c) throws IOException
```

Métodos heredados de java.io.Writer

```
void write(char[] cbuf) throws IOException
```

```
void write(string str) throws IOException
```

```
void write(string str, int off, int len) throws IOException
```

Métodos heredados de la clase java.lang.Object:

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait, wait.

Clase: PipedReader

Constructor:

```
public PipedReader(PipedWriter s) throws IOException
```

Crea un flujo lector conectado a uno escritor.

```
public PipedReader()
```

Métodos:

```
void close() throws IOException
```

```
void connect(PipedWriter s) throws IOException
```

Conecta un conducto lector a uno escritor. Si el objeto ya está conectado a algún otro escritor, lanza la excepción.

```
void read() throws IOException
```

```
void write(char[] cbuf, int off, int len) throws IOException
```

```
boolean ready() throws IOException
```

Métodos heredados de java.io.Reader

```
public void mark(int t) throws IOException
```

```
public boolean markSupported()
```

```
public int read(char[] cbuf) throws IOException
```

```
public void reset() throws IOException
```

```
public long skip(long n) throws IOException
```

Métodos heredados de la clase java.lang.Object:

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait, wait.

Nota: Si se intenta utilizar un flujo Piped antes de que esté conectado, o se intenta conectar cuando ya está conectado, se produce una IOException.

Solución del Ejercicio 1

```
/*
 * Ejercicio1.java
 *
 * Solución al Ejercicio 1 del examen de Junio de 2002 de
 * Fundamentos de Telemática.
 */

import java.io.*;

public class Ejercicio1 {

    public static void main(String[] args) {
        /*Definimos out tipo PipedWriter*/
        PipedWriter out = new PipedWriter();

        /*Definimos in tipo PipedReader*/
        PipedReader in = new PipedReader(out);

        /*Instanciamos los hilos*/
        GeneradorTexto datos = new GeneradorTexto (out);
        HiloLector lectura = new HiloLector (in);

        /*Comenzamos la ejecución de los hilos*/
        datos.start();
        lectura.start();

        try{
            while(datos.tt.isAlive() || lectura.t.isAlive())
                Thread.currentThread().sleep(1000);
        }catch(InterruptedException e){
        }
        /*
        try{
            datos.tt.join();
            lectura.t.join();
        }catch(InterruptedException e){
        }
        */
        System.out.println("fin");
    }
}

/**
 * Clase GeneradorTexto hereda de Thread y contiene el método run()
 */
class GeneradorTexto extends Thread {
    private Writer out;
    Thread tt;

    GeneradorTexto (Writer out){
        this.out = out;
        tt = new Thread(this);
    }

    public void run() {
        try{
            try{
                for (;;)
                    out.write(System.in.read());
            /*
            for (char c='a'; c<='z'; c++)

```

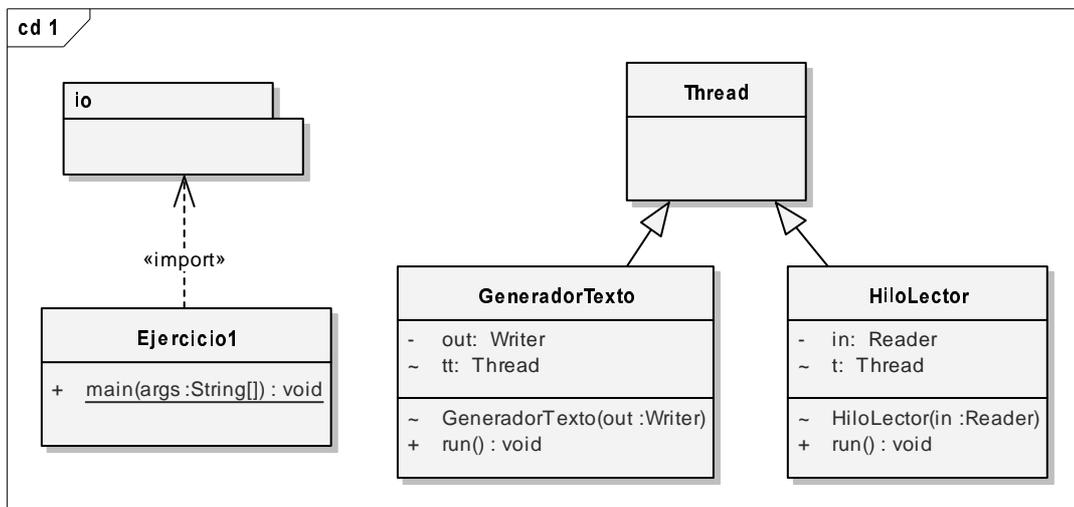
```

        out.write(c);
        */
    }catch(IOException e){
        getThreadGroup().uncaughtException(this,e);
    }finally{
        out.close();
    }
    }catch(Exception e){
    }
}

/**
 * Clase HiloLector hereda de Thread y contiene el método run()
 */
class HiloLector extends Thread {
    private Reader in;
    Thread t;
    HiloLector (Reader in){
        this.in = in;
        t = new Thread(this);
    }

    public void run() {
        int cr;
        try {
            try{
                while((cr = in.read()) != -1)
                    System.out.println((char)cr);
            }catch(IOException e){
                getThreadGroup().uncaughtException(this,e);
            }finally{
                in.close();
            }
        }catch(Exception e){
        }
    }
}
}

```



junio02_1

Class Ejercicio1

java.lang.Object

└─ junio02_1.Ejercicio1

```
public class Ejercicio1
    extends java.lang.Object
```

Solución al Ejercicio 1 del examen de Junio de 2002 de Fundamentos de Telemática.

Constructor Summary

Ejercicio1()	
------------------------------	--

Method Summary

static void	main (java.lang.String[] args)	Comprueba argumentos de la línea de comandos
-------------	--	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ejercicio1

```
public Ejercicio1()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Comprueba argumentos de la línea de comandos

Parameters:

args - Argumentos de la línea de comandos

Ejercicio 2 (21/06/2002) applet, awt.

En este ejercicio se implementa un applet en una ventana con una figura moviéndose por el área de trabajo. La figura es una pelota roja (un círculo) que se dibujará con el método `fillOval()` (*Ver información sobre el método*). El área de trabajo es la ventana del applet.

Las ventanas son contenedores de información donde se dibuja con los métodos gráficos del AWT. Estas ventanas pueden ser, o bien la ventana principal de una applet, o bien una ventana hija de una applet o bien la ventana de una aplicación independiente. La posición (0,0) de cada ventana corresponde con la esquina superior izquierda. Las coordenadas se especifican en píxeles. Todas las salidas que se realizan sobre una ventana tienen lugar a través de un *contexto gráfico*. Un contexto gráfico está encapsulado en la clase `Graphics` y se puede obtener de dos maneras:

- ✓ Se pasa a un applet cuando se llama alguno de sus métodos, como `paint()` o `update()`.
- ✓ Lo devuelve el método `getGraphics()` (*Ver información sobre el método*).

Cuando la pelota alcanza los contornos de la ventana del applet, rebota. El movimiento de la pelota se realizará mediante el uso de hilos. El movimiento se realiza incrementando en 1 los ejes X e Y. El radio de la pelota es 10, y el incremento en la posición cada vez que se actualiza es de 1 en las dos direcciones. Se pide:

1.- Especifique el código de toda la clase. Implemente los métodos:

- ✓ `public void init()` (que determina las dimensiones del applet (método `getSize().width` y `getSize().height`) y da una posición inicial (elegida por el programador) de partida para la pelota, y el color de fondo blanco.
- ✓ `public void start()`, que realiza la instanciación del hilo y arranca su ejecución.
- ✓ `public void stop()`, que para la ejecución del hilo.
- ✓ `public void paint(Graphics g)`.
- ✓ `public void run()`, que en un bucle infinito llama al método `mover()`.
- ✓ `void mover()` que actualice la posición de la pelota.

Tal cual, el código anterior presenta problemas. La pelota se mueve rápidamente y produce un molesto parpadeo causado por el borrado de la ventana del applet y vuelta a pintarlo de nuevo en una nueva posición. El método `repaint` no llama directamente a `paint` sino a `update`, que borra la ventana y a continuación se llama a `paint`. Una solución es redefiniendo el método `update` con la llamada a `paint`.

2.- Implemente esta solución redefiniendo el método `update` con la llamada a `paint`.

La solución es parcial, y se propone una nueva solución conocida como la técnica del *doble buffer*. Con la técnica de *doble almacenamiento temporal de gráficos (o doble buffer)*, mientras el programa exhibe una imagen en la pantalla puede construir la siguiente imagen en un *buffer* fuera de la pantalla. Luego, cuando llega el momento de exhibir esa siguiente imagen, se puede colocar en la pantalla de inmediato.

Para implementar esta técnica hay que:

- ✓ Crear un objeto `Image` en blanco, que es el lugar donde se almacenarán los píxeles que se van a exhibir. Se crea con el método `createImage()`.
- ✓ Dibujar en ese objeto usando métodos de la clase `Graphics`. Toda imagen tiene asociado un contexto gráfico, es decir, un objeto de la clase `Graphics` que hace posible dibujar. Para obtener el contexto gráfico.

3.- Implemente esta solución redefiniendo el método `update` para implementar la técnica del *doble buffer*.

Nota 1: Para actualizar la información mostrada en la ventana de un applet, se llama al método `void repaint()`.

Nota 2: `ImageObserver` es una interfaz implementada por la clase `Component`. Este argumento es importante cuando se exhiben imágenes grandes que tardan mucho tiempo en descargarse de Internet. En el ejercicio no se usa y se deberá poner a `null`.

Nota 3: `Image` es una clase superclase de todas las clases que representan imágenes gráficas.

Algunos métodos de la clase `Graphics` del paquete `java.awt`:

`void fillOval (int x, int y, int ancho, int alto)`

Para dibujar una elipse rellena se utiliza el método `fillOval()`. La elipse se dibuja dentro del rectángulo cuya esquina superior izquierda corresponde a la posición `x`, `y` y cuyas dimensiones son las especificadas por `ancho` y `alto`. Para dibujar un círculo basta con que las dimensiones especificadas correspondan a las de un cuadrado en lugar de a las de un rectángulo.

`Dimension getSize()`

Para obtener las dimensiones de la ventana del applet se puede utilizar el método `getSize()`. Este método devuelve un objeto de la clase `Dimension`. La clase `Dimension` encapsula en un objeto las componentes (variables de instancia) `width` y `height` como enteros.

`void setColor(Color c)`

Para configurar el color del gráfico se utiliza el método `setColor()` de la clase `Graphics` del paquete `java.awt`.

Para configurar el color rojo, `static Color red`.

Para configurar el color blanco, `static Color white`.

`void fillRect(int x, int y, int width, int height)`

Llena el rectángulo.

`boolean drawImage(Image img, int x, int y, ImageObserver observer)`

Es un método sobrecargado y sólo se da información de la versión útil para este ejercicio.

Dibuja la imagen tal y como esté disponible en ese momento. `x` e `y` deben ser 0 en el ejercicio.

De la clase `Image` del `java.awt`

`Graphics getGraphics()`

Crea un contexto gráfico para dibujar una imagen fuera de pantalla.

De la clase `Component` del `java.awt`

`Graphics getBackground()`

Obtiene el color de fondo.

`Graphics createImage(int anchoApplet, int altoApplet)`

Crea una imagen fuera de pantalla para usarla en el doble buffer.

Solución del Ejercicio 2

```
/*
 * AnimaApplet1.java
 *
 * Solución al Ejercicio 2 del examen de Junio de 2002 de
 * Fundamentos de Telemática.
 */

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

public class AnimaApplet1 extends Applet implements Runnable{ //Apartado 1
    Thread anima;
    int radio = 10;
    int x, y;
    int dx = 1;
    int dy = 1;
    int anchoApplet;
    int altoApplet;

    public void init(){
        setBackground(Color.white);

        anchoApplet = getSize().width;
        altoApplet = getSize().height;

        x = anchoApplet/4;
        y = altoApplet/2;
    }

    public void start(){
        if(anima == null){
            anima = new Thread(this);
            anima.start();
        }
    }

    public void stop(){
        if(anima != null){
            this.stop();
            anima = null;
        }
    }

    public void run(){
        while(true){
            mover();
        }
    }

    void mover(){
        x += dx;
        y += dy;
        if(x >= (anchoApplet-radio) || x < radio) dx *= -1;
        if(y >= (altoApplet-radio) || y < radio) dy *= -1;
        repaint(); //llama a paint. Ver Nota 1
    }
}
```

```

public void paint(Graphics g){
    g.setColor(Color.red);
    g.fillOval(x-radio, y-radio, 2*radio, 2*radio);
}
}

public void update(Graphics g){                                     //Apartado 2
    paint(g);
}

                                                                    //Apartado 3
int anchoApplet;
int altoApplet;
Image imagen;
Graphics gBuffer;

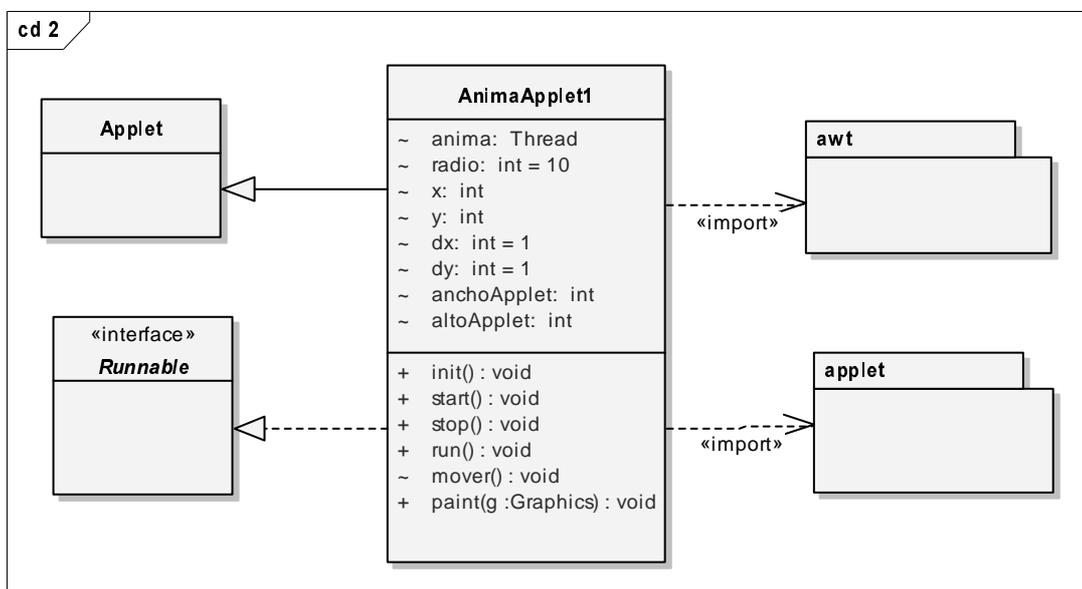
public void update(Graphics g){
    if(gBuffer == null){
        Image imagen = createImage(anchoApplet, altoApplet);
        gBuffer      = imagen.getGraphics();
    }

    gBuffer.setColor(getBackground());
    gBuffer.fillRect(0,0, anchoApplet, altoApplet);

    gBuffer.setColor(Color.red);
    gBuffer.fillOval(x-radio, y-radio, 2*radio, 2*radio);

    g.drawImage(imagen, 0, 0, null);
}
}

```



junio02_2

Class AnimaApplet1

```
java.lang.Object
├── java.awt.Component
│   ├── java.awt.Container
│   │   ├── java.awt.Panel
│   │   │   ├── java.applet.Applet
│   │   │   └── junio02_2.AnimaApplet1
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.lang.Runnable, javax.accessibility.Accessible

```
public class AnimaApplet1
    extends java.applet.Applet
    implements java.lang.Runnable
```

Solución al Ejercicio 2 del examen de Junio de 2002 de Fundamentos de Telemática.

See Also:

[Serialized Form](#)

Nested Class Summary

Nested classes/interfaces inherited from class java.applet.Applet

java.applet.Applet.AccessibleApplet

Nested classes/interfaces inherited from class java.awt.Panel

java.awt.Panel.AccessibleAWTPanel

Nested classes/interfaces inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

Nested classes/interfaces inherited from class java.awt.Component

java.awt.Component.AccessibleAWTComponent,
java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

AnimaApplet1()

Method Summary

void init()

void paint(java.awt.Graphics g)

void run()

void start()

void stop()

Methods inherited from class java.applet.Applet

destroy, getAccessibleContext, getAppletContext, getAppletInfo, getAudioClip, getAudioClip, getCodeBase, getDocumentBase, getImage, getImage, getLocale, getParameter, getParameterInfo, isActive, newAudioClip, play, play, resize, resize, setStub, showStatus

Methods inherited from class java.awt.Panel

addNotify

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

AnimaApplet1

public AnimaApplet1()

Method Detail

init

public void init()

Overrides:

init in class java.applet.Applet

start

public void start()

Overrides:

start in class java.applet.Applet

stop

public void stop()

Overrides:

stop in class java.applet.Applet

run

public void run()

Specified by:

run in interface java.lang.Runnable

paint

public void paint(java.awt.Graphics g)

Overrides:

paint in class java.awt.Container

[Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Ejercicio 3 (21/06/2002) polimorfismo, herencia.

En este ejercicio se implementa parte de un sistema de control aéreo en Java, en el que se va a utilizar la selección dinámica de método. Para ello, se declaran un conjunto de clases que van a representar los aviones detectados por el sistema.

Se pide:

1.- Implementar una clase abstracta que contenga cinco atributos (variables de instancia) de tipo **double** para presentar la longitud, latitud, altura, velocidad y dirección del avión; y un método abstracto **ver.** (**void ver()**), este método presentará por pantalla toda la información disponible del apartado).

2.- Implementar una nueva clase no abstracta **OV**, derivada de la anterior, que se utilizará para representar los aviones no identificados. Esta clase añade una nueva variable, un entero estático, que contendrá el número de objetos creados por el sistema. Se implementarán dos constructores: uno sin parámetros y otro con cinco parámetros, correspondientes a los atributos definidos por la clase abstracta.

3.- Para representar los aviones identificados, implementar la clase **Aeronave**, que deriva de **OV** y no debe permitir herencias posteriores y que:

- ✓ Añade dos atributos que no deben ser accedidos desde ningún otro lugar fuera de esta clase:
 - ✓ Un entero que se usa como identificador de la clase y
 - ✓ Un real de precisión doble que representa la autonomía de la nave.
- ✓ Se deben implementar dos constructores:
 - ✓ Uno con siete parámetros (longitud, latitud, altura, velocidad, dirección, identificador y autonomía) usando la llamada al constructor de la superclase, y
 - ✓ Otro sin parámetros.
- ✓ Además se definirá el método **Asignación** que asignará el valor de único parámetro al atributo identificador o autonomía, en función del tipo de dicho parámetro.
- ✓ Se definirán dos métodos más que devolverán los valores de estos dos atributos.
- ✓ Como último método de esta clase, se habrá de sobrescribir el método de visualización para que muestre también el valor de los dos nuevos atributos.

4.- Explique **BREVEMENTE** en qué consiste la selección dinámica de método y declare una clase con el método principal **main**, donde se aplique.

5.- ¿Qué sucede cuando al método de visualización se le aplica el modificador **final** en la clase **OV**?

Solución del Ejercicio 3

```
/*
 * SelMetDin.java
 *
 * Solución del Ejercicio 3 del examen de Junio de 2002 de
 * Fundamentos de Telemática.
 */

class SelMetDin{
    public static void main(String args[]){
        ClaseAbstracta var0;
        OV          var1 = new OV();
        Aeronave     var2 = new Aeronave();

        var1.ver();
        var2.ver();
        var2.Asignacion(3);
        var2.Asignacion(3.);
        System.out.println();
        var0 = var1;
        var0.ver();
    }
}

/**
 * Clase Abstracta
 */
abstract class ClaseAbstracta{
    double longitud;
    double latitud;
    double altura;
    double velocidad;
    double direccion;

    abstract void ver();
}

/**
 * Clase OV hereda de ClaseAbtracta
 */
class OV extends ClaseAbstracta{
    static int NumObjCreados = 0;
    OV(){
    OV(double lng, double lat, double alt, double vel, double dir){
        super.longitud = lng;
        super.latitud  = lat;
        super.altura   = alt;
        super.velocidad = vel;
        super.direccion = dir;
    }
    void ver(){
        System.out.println("La longitud = " + longitud);
        System.out.println("La latitud  = " + latitud);
        System.out.println("La altura   = " + altura);
        System.out.println("La velocidad = " + velocidad);
        System.out.println("La dirección = " + direccion);
    }
}

/**
```

```

* Clase Aeronave hereda de OV y no permite herencias (final)
*/
final class Aeronave extends OV{
    private int IdentifClase;
    private double autonomia;

    Aeronave (){}

    Aeronave(double lng, double lat, double alt, double vel, double dir,
    int id, double aut){
        super(lng, lat, alt, vel, dir);
        this.IdentifClase = id;
        this.autonomia     = aut;
    }

    void Asignacion(int valor){
        IdentifClase = valor;
        System.out.println("Asignación de valor Entero ");
    }
    void Asignacion(double valor){
        autonomia = valor;
        System.out.println("Asignación de valor Real ");
    }
    int ObtieneIdClas(){
        return IdentifClase;
    }
    double ObtieneAutoClas(){
        return autonomía;
    }
    void ver(){
        System.out.println("La longitud      = " + longitud );
        System.out.println("La latitud      = " + latitud );
        System.out.println("La altura      = " + altura );
        System.out.println("La velocidad   = " + velocidad);
        System.out.println("La dirección   = " + direccion);
        System.out.println("La IdentifClase = " + IdentifClase);
        System.out.println("La autonomía   = " + autonomía );
    }
}

```

/* Solucion del apartado 4

*

* La selección dinámica de método es un mecanismo mediante el cual una llamada a un método se resuelve en tiempo de ejecución en función del número y tipo de los parámetros pasados como argumentos.

*/

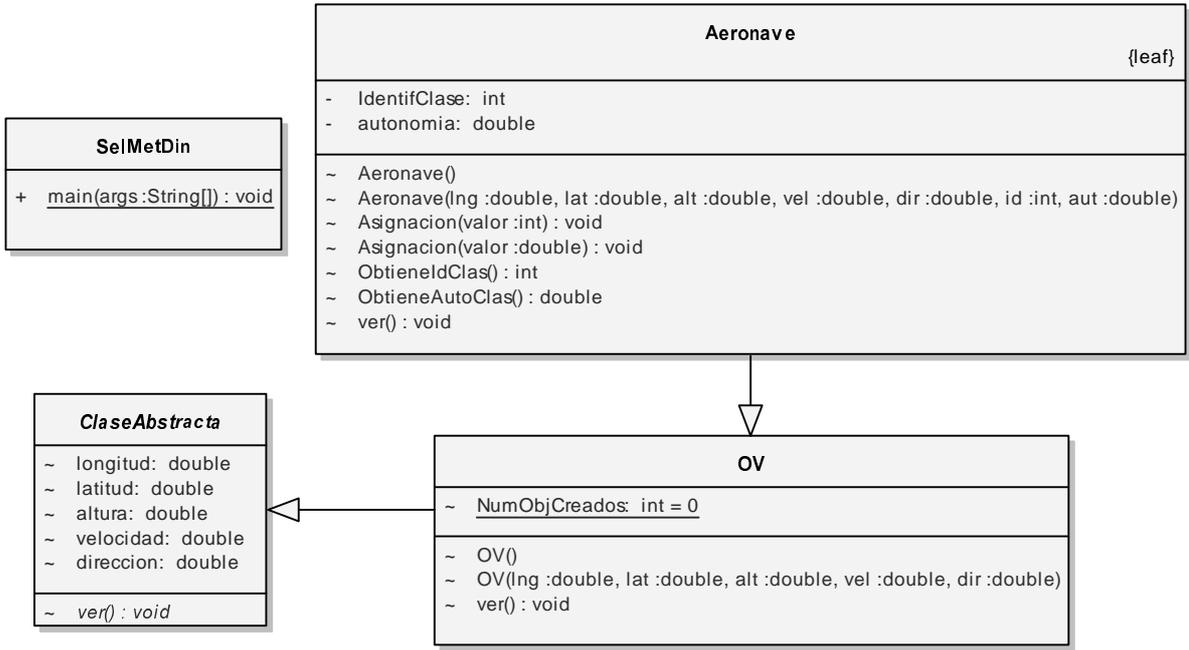
/* Solucion del apartado 5

*

* El modificador final en el método indicado imposibilita la sobre-escritura de éste en cualquier clase heredera.

*/

cd 3



junio02_3

Class SelMetDin

java.lang.Object

└─ junio02_3.SelMetDin

```
public class SelMetDin
    extends java.lang.Object
```

Solución del Ejercicio 3 del examen de Junio de 2002 de Fundamentos de Telemática.

Constructor Summary

SelMetDin()	
-----------------------------	--

Method Summary

static void	main (java.lang.String[] args)
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SelMetDin

```
public SelMetDin()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Ejercicio 1 (10/9/2002) clases anidadas

Las clases e interfaces se pueden declarar dentro de otras clases e interfaces (como miembros o en bloques de código). Esta declaración se conoce como *clases anidadas* o *tipos anidados*. Estas clases son miembros de otras clases, y su visibilidad y alcance dependerán, al igual que los métodos y las variables de instancia de la clase, de la forma en que estén declaradas.

La posibilidad de definir tipos anidados tiene dos fines principales:

- Permite estructurar los tipos y su alcance en grupos relacionados lógicamente.
- Se pueden utilizar para conectar objetos relacionados lógicamente de forma simple y efectiva (como por ejemplo los usados en los eventos del AWT, y en la arquitectura de componentes JavaBeans).

Las clases anidadas pueden ser estáticas o no estáticas. Al igual que los métodos y variables estáticas una clase anidada estática está asociada con una clase a la que pertenece. Para utilizar los métodos y variables de una clase estática se deberá hacer mediante la instanciación de la clase anidada.

En este problema se declararán dos clases (**Permisos**, que será una clase estática, y **Accion**, que no será estática) dentro de otra clase (**CuentaBancaria**). Cada objeto **CuentaBancaria** mantendrá la información del número de cuenta (**numero**), del balance (**balance**), y una última acción realizada (**UltimaAccion**); esta información será privada (**private**) a la clase **CuentaBancaria**.

1. (1 pto) Declare la clase **Permisos** dentro de la clase **CuentaBancaria** con tres variables de instancia de tipo **boolean**: **puedeDepositar**, **puedeRetirar**, y **puedeCerrar**.
2. (3 ptos) Implemente el método **permisosDe** de tal forma que se pueda escribir el código que aparece a continuación, fuera de la clase **CuentaBancaria**:

```
CuentaBancaria cuenta = new CuentaBancaria();  
CuentaBancaria.Permisos perm = cuenta.permisosDe();
```

Este método pone a **true** las variables declaradas en el apartado anterior y las saca por pantalla.

Las clases anidadas no estáticas se denominan *clases internas*. Los miembros no estáticos de una clase se asocian con instancias de la clase: los campos no estáticos son variables de instancia y los métodos no estáticos operan sobre una instancia. De igual forma, una *clase interna* se asocia también con una instancia de la clase. Más concretamente, un objeto de una clase interna se asocia siempre con un objeto de la clase que la incluye: el *objeto que lo incluye*.

3. (4 ptos) Declare la clase **Accion** que almacena una sola acción sobre la cuenta, y que está declarada dentro de **CuentaBancaria**. Esta clase contiene dos variables privadas, el número de la cuenta y la cantidad de la última operación realizada. Implemente el constructor para inicializar ambas variables y sobrescriba el método **toString** de tal forma que presenta dicha información por pantalla.
4. (1 pto) Implemente el método **deposito** de tal forma que pasándole como parámetro una cantidad se actualiza la correspondiente variable de instancia y guarda la última acción realizada.
5. (1 pto) Implemente el método **reintegro** de tal forma que pasándole como parámetro una cantidad se actualiza la correspondiente variable de instancia y guarda la última acción realizada.

Solución del Ejercicio1

```
/*
 * SeptInterfaz.java
 *
 * Solución al Ejercicio 1 del examen de Septiembre de 2002 de
 * Fundamentos de Telemática.
 */

/**
 * Mantiene la información del número de cuenta, del balance y una última
acción
 * realizada.
 */
class CuentaBancaria{
    private long numero;
    private long balance;
    private Accion ultimaAcc;

    /** Clase que contiene 3 variables de tipo boolean*/
    public static class Permisos {
        public boolean puedeDepositar;
        public boolean puedeRetirar;
        public boolean puedeCerrar;
    }
    /**
     * Sacar por pantalla los valores de las 3 variables de instancia de un
     * objeto
     * de la clase Permisos.
     * @return objeto de la clase Permisos inicializado a true
     */
    public Permisos permisosDe(){
        Permisos pl = new Permisos();
        pl.puedeDepositar=true;
        pl.puedeRetirar=true;
        pl.puedeCerrar=true;
        System.out.println(pl.puedeDepositar);
        System.out.println(pl.puedeRetirar);
        System.out.println(pl.puedeCerrar);
        return pl;
    }
    /**Almacena una acción sobre la cuenta. Contiene dos variables
privadas*/
    public class Accion {
        private String cuenta;
        private long cantidad;
        /**Inicializa las variables mediante los parámetros que recibe*/
        Accion(String cuenta, long cantidad){
            this.cuenta = cuenta;
            this.cantidad = cantidad;
        }
        /**Imprime la información generada en la clase*/
        public String toString(){
            return (numero + ": "+cuenta+ " "+cantidad);
        }
    }
    public void deposito(long cantidad){
        balance += cantidad;
        ultimaAcc = new Accion("deposito", cantidad);
    }
    public void reintegro(long cantidad){
        balance -= cantidad;
    }
}
```

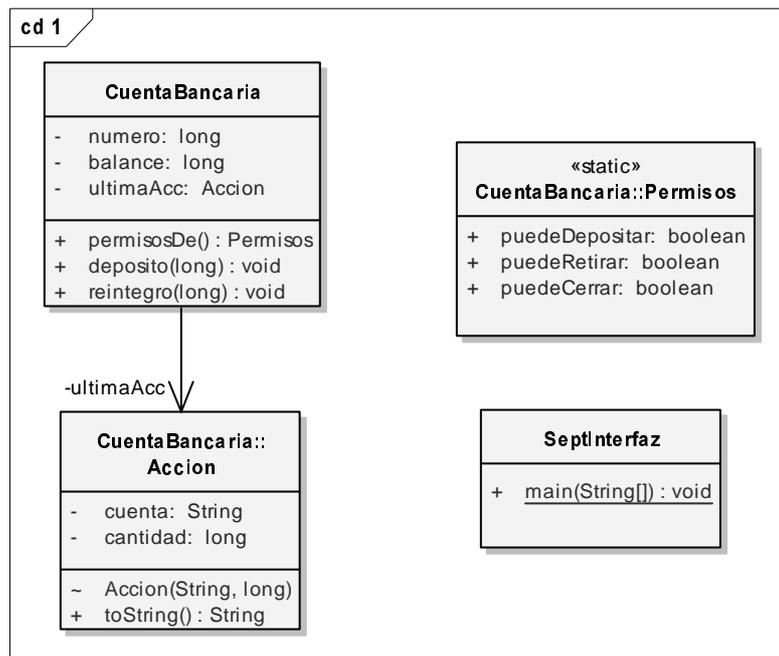
```

        ultimaAcc = new Accion("reintegro", cantidad);
    }
}

public class SeptInterfaz{

    /**
     * Crea un objeto de la clase CuentaBancaria y una referencia al
     * objeto * de clase Permisos que hay dentro, y lo inicializa y
     * saca por pantalla * mediante el método permisosDe.
     */
    public static void main(String args[]){
        CuentaBancaria cuenta = new CuentaBancaria();
        CuentaBancaria.Permisos perm = cuenta.permisosDe();
    }
}

```



Class *SeptInterfaz*

java.lang.Object

└─ SeptInterfaz

```
public class SeptInterfaz
    extends java.lang.Object
```

Solución al Ejercicio 1 del examen de Septiembre de 2002 de Fundamentos de Telemática.

Constructor Summary

[SeptInterfaz](#)()

Method Summary

static void [main](#)(java.lang.String[] args)

Crea un objeto de la clase CuentaBancaria y una referencia al objeto * de clase Permisos que hay dentro, y lo inicializa y saca por pantalla * mediante el método permisosDe.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SeptInterfaz

```
public SeptInterfaz()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Crea un objeto de la clase CuentaBancaria y una referencia al objeto * de clase Permisos que hay dentro, y lo inicializa y saca por pantalla * mediante el método permisosDe.

Ejercicio 2 (10/9/2002) hilos, intérprete

El parking de un ferry dispone de varias puertas de acceso (dos en este ejercicio). Cuando un automovilista accede por una puerta de entrada, recoge un ticket que abonará en destino. El procedimiento de carga del barco consiste simplemente en dejar pasar automóviles por sus puertas de acceso hasta que el parking esté completo, momento en el cual zarpa del puerto origen.

El sistema de control de carga está compuesto por una plataforma LINUX sobre la que se tiene: un proceso de control por cada puerta, un contador global representado por una variable entera denominada "cuenta", y un proceso de monitorización de este contador, y cuya función es detectar cuándo se ha llenado el parking. El siguiente código es una aproximación a este problema.

```
class Cuenta{
    private int valor;
    Cuenta(){
        valor = 0;
    }

    /*synchronized*/ public void
        setValor(int valor){
        this.valor = valor;
    }

    public int getValor(){
        return valor;
    }
}

class Puerta extends Thread{
    private Cuenta cuenta;
    private int N;

    Puerta(Cuenta cuenta, int N){
        this.cuenta = cuenta;
        this.N = N;
    }

    public void run(){
        try{
            /*synchronized (cuenta)*/ {
                int v = 0;
                for(int i=0; i<N;i++){
                    v = cuenta.getValor();
                    System.out.println("Valor "+v);

                    sleep(40);

                    v++;
                    sleep(40);
                    cuenta.setValor(v);
                }
            }catch(InterruptedException e){}
        }
    }
}

class SeptHilos {
    public static void main(String args[]){
        Cuenta cuenta = new Cuenta();
        Puerta puertal = new
        Puerta(cuenta,20);
        Puerta puerta2 = new
        Puerta(cuenta,20);

        puertal.start();
        puerta2.start();

        try{
            puertal.join();
            puerta2.join();
        }catch(InterruptedException
e){
        System.out.println("Excepción: " + e);
        }
        System.out.print("Numero coches =
");
        System.out.println(cuenta.getValor()
);
    }
}
```

1. ¿Qué resultado se produce (por pantalla) en la interpretación del código Java que aparece arriba?
2. Si se cambia solamente la línea `/*synchronized*/ public void setValor(int valor){` por la línea `synchronized public void setValor(int valor){` cuál es el resultado que produce la interpretación del código Java.
3. Si se cambia solamente la línea `/*synchronized (cuenta)*/ {` por la línea `synchronized (cuenta) {` cuál es el resultado que produce la interpretación del código Java.
4. Si se producen los cambios realizados en el apartado 2 y 3 a la vez, cuál es el resultado que se produce en la interpretación del código Java.
5. Si se cambia solamente la línea `class Cuenta{` por la línea `synchronized class Cuenta{` cuál es el resultado que produce la interpretación del código Java.

Ejercicio2 (apartados 1 y 2)

Valor 0
Valor 0
Valor 1
Valor 1
Valor 2
Valor 2
Valor 3
Valor 3
Valor 4
Valor 4
Valor 5
Valor 5
Valor 6
Valor 6
Valor 7
Valor 7
Valor 8
Valor 8
Valor 9
Valor 9
Valor 10
Valor 10
Valor 11
Valor 11
Valor 12
Valor 12
Valor 13
Valor 13
Valor 14
Valor 14
Valor 15
Valor 15
Valor 16
Valor 16
Valor 17
Valor 17
Valor 18
Valor 18
Valor 19
Valor 19
Numero final de coches = 20

El resultado del apartado 1 y el del 2 son iguales

Ejercicio2 (apartados 3 y 4)

Valor 0
Valor 1
Valor 2
Valor 3
Valor 4
Valor 5
Valor 6
Valor 7
Valor 8
Valor 9
Valor 10
Valor 11
Valor 12
Valor 13
Valor 14
Valor 15
Valor 16
Valor 17
Valor 18
Valor 19
Valor 20
Valor 21
Valor 22
Valor 23
Valor 24
Valor 25
Valor 26
Valor 27
Valor 28
Valor 29
Valor 30
Valor 31
Valor 32
Valor 33
Valor 34
Valor 35
Valor 36
Valor 37
Valor 38
Valor 39
Numero final de coches = 40

El resultado del apartado 3 y el del 4 son iguales

Ejercicio2 (apartado 5)

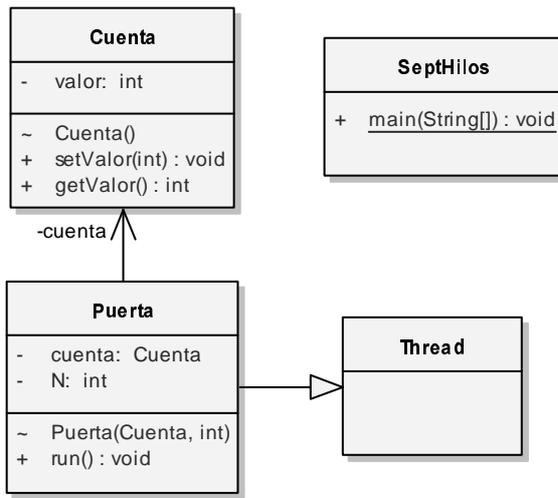
SeptHilos.java:1: Class or interface declaration expected.

```
synchronized class Cuenta{
```

```
^
```

1 error

cd 2



Class *SeptHilos*

java.lang.Object
└─ **SeptHilos**

```
public class SeptHilos  
extends java.lang.Object
```

Solución al Ejercicio 2 del examen de Septiembre de 2002 de Fundamentos de Telemática.

Constructor Summary

SeptHilos()	
-----------------------------	--

Method Summary

static void main (java.lang.String[] args)
--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SeptHilos

```
public SeptHilos()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Ejercicio 3 (10/9/2002) interfaz, polimorfismo, paquetes, interprete

En este ejercicio se usan las interfaces de Java. Se pretende declarar las clases **Vector2D** y **Vector3D**, en el paquete **PaqueteVector**. Estas clases contendrán como atributos los números reales necesarios para representar vectores en dos y tres dimensiones respectivamente.

Para manejar ambos tipos de datos de un modo unificado, ambas clases implementarán la interfaz **IVector**, que estará en el paquete **paqueteIVector** e incorporará lo siguiente:

- Un método que muestre en pantalla el valor del vector: **ver**.
- Un valor constante **PI**.

Las clases añadirán los siguientes métodos (que no estarán reflejados en la interfaz):

- Un constructor con parámetros y otro sin parámetros.
- Método sin modificador de acceso que sume dos vectores de la misma clase: **suma**.

1. Implemente la interfaz (2p) y las clases especificadas (3p). Determine los ficheros, directorios y órdenes de compilación para el código escrito.
2. (3p) Las interfaces admiten la resolución dinámica de método durante la ejecución. En el paquete por defecto y sin nombre implemente el método **main**, de tal forma, que se aplique la resolución dinámica de método.
3. (2p) Presente por pantalla en el método anterior (**main**) el valor de **PI**.

Solución del Ejercicio3

```
javac ./paqueteIVector/IVector.java
```

```
package paqueteIVector;

public interface IVector {
    void ver();
    final static double PI = 3.14;
}
```

```
javac ./PaqueteVector/Vector2D.java
```

```
package PaqueteVector;
import paqueteIVector.*;

public class Vector2D implements IVector {
    double x1;
    double x2;
    public Vector2D(){}
    public Vector2D(double x1, double x2){
        this.x1 = x1;
        this.x2 = x2;
    }
    public void ver(){
        System.out.println("x1 = " + this.x1);
        System.out.println("x2 = " + this.x2);
    }
    public Vector2D suma(Vector2D v1,Vector2D v2){
        Vector2D aux = new Vector2D();
        aux.x1 = v1.x1 + v2.x1;
        aux.x2 = v1.x2 + v2.x2;
        return aux;
    }
}
```

javac ./PaqueteVector/Vector3D.java

```
package PaqueteVector;
import paqueteIVector.*;

public class Vector3D implements IVector {
    double x1;
    double x2;
    double x3;
    public Vector3D(){
    public Vector3D(double x1, double x2, double x3){
        this.x1 = x1;
        this.x2 = x2;
        this.x3 = x3;
    }
    public void ver(){
        System.out.println("x1 = " + this.x1);
        System.out.println("x2 = " + this.x2);
        System.out.println("x3 = " + this.x3);
    }
    public Vector3D suma(Vector3D v1,Vector3D v2){
        Vector3D aux = new Vector3D();
        aux.x1 = v1.x1 + v2.x1;
        aux.x2 = v1.x2 + v2.x2;
        aux.x3 = v1.x3 + v2.x3;
        return aux;
    }
}
```

javac Interfaz.java

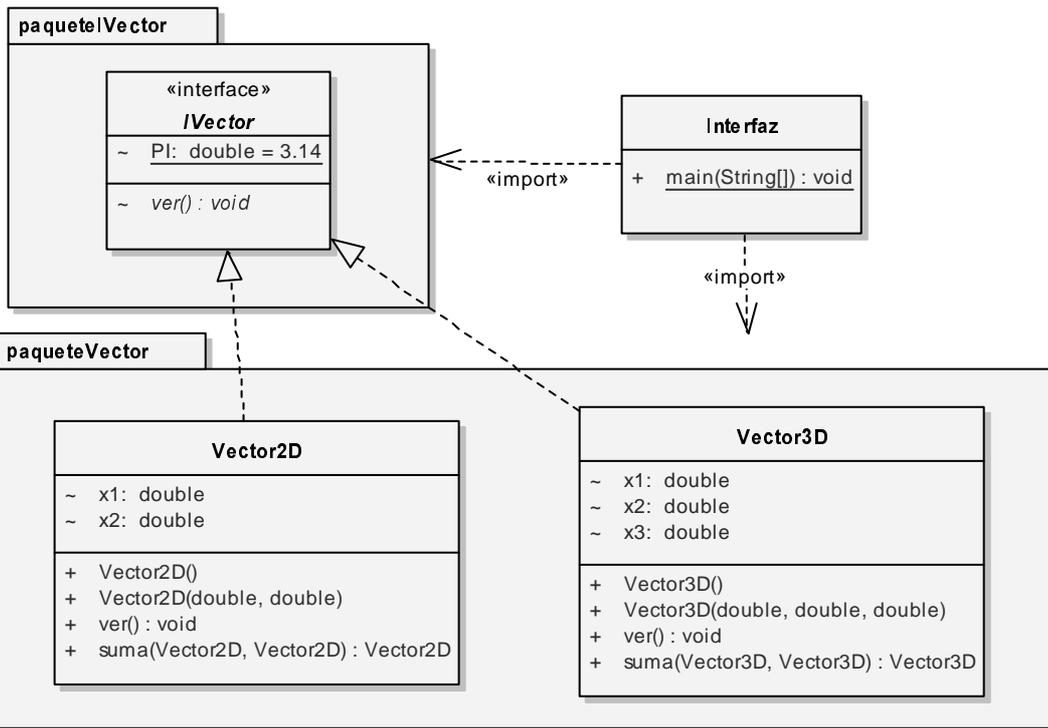
```
/**En el paquete por defecto y sin nombre.*/
import PaqueteVector.*;
import paqueteIVector.*;

class Interfaz{
    public static void main(String args[]){
        IVector variable;
        Vector2D obj1 = new Vector2D(3.0,3.0);
        Vector3D obj2 = new Vector3D(3.0,3.0,3.0);

        variable = obj1;
        variable.ver();

        variable = obj2;
        variable.ver();

        System.out.println("La constante
"+IVector.PI);
    }
}
```



[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

Packages

[paquetelVector](#)

[PaqueteVector](#)

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

Package *paquetelVector*

Interface Summary

[IVector](#)

paquetelVector

Interface IVector

All Known Implementing Classes:

[Vector2D](#), [Vector3D](#)

public interface **IVector**

Field Summary

static double **PI**

Method Summary

void **ver** ()

Field Detail

PI

public static final double **PI**

See Also:

[Constant Field Values](#)

Method Detail

ver

```
public void ver()
```

Package PaqueteVector

Class Summary

Vector2D	
Vector3D	

PaqueteVector

Class Vector2D

java.lang.Object

PaqueteVector.Vector2D

All Implemented Interfaces:

[IVector](#)

```
public class Vector2D
extends java.lang.Object
implements IVector
```

Field Summary

(package private) double	<u>x1</u>
(package private) double	<u>x2</u>

Fields inherited from interface paqueteIVector.[IVector](#)

[PI](#)

Constructor Summary

Vector2D ()	
Vector2D (double x1, double x2)	

Method Summary

	suma (Vector2D v1, Vector2D v2)
--	--

<u>Vector2D</u>	
void	<u>ver</u> ()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

x1

double x1

x2

double x2

Constructor Detail

Vector2D

public **Vector2D**()

Vector2D

public **Vector2D**(double x1,
double x2)

Method Detail

ver

public void **ver**()

Specified by:

ver in interface IVector

suma

public Vector2D **suma**(Vector2D v1,
Vector2D v2)

PaqueteVector

Class Vector3D

java.lang.Object

PaqueteVector.Vector3D

All Implemented Interfaces:

IVector

public class **Vector3D**
extends java.lang.Object
implements IVector

Field Summary

(package private) double	<u>x1</u>
-----------------------------	-----------

(package private) double	<u>x2</u>
-----------------------------	-----------

(package private) double	<u>x3</u>
-----------------------------	-----------

Fields inherited from interface paqueteIVector. IVector

PI

Constructor Summary

Vector3D ()

Vector3D(double x1, double x2, double x3)

Method Summary

<u>Vector3D</u>	<u>suma</u> (<u>Vector3D</u> v1, <u>Vector3D</u> v2)
-----------------	---

void	<u>ver</u> ()
------	---------------

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

x1

double x1

x2

double x2

x3

double x3

Constructor Detail

Vector3D

public **Vector3D**()

Vector3D

```
public Vector3D(double x1,  
                double x2,  
                double x3)
```

Method Detail

ver

```
public void ver()
```

Specified by:

ver in interface IVector

suma

```
public Vector3D suma(Vector3D v1,  
                    Vector3D v2)
```

Class Interfaz

java.lang.Object

Interfaz

class **Interfaz**

extends java.lang.Object

Constructor Summary

(package private)	<u>Interfaz</u> ()
-------------------	--------------------

Method Summary

static void	<u>main</u> (java.lang.String[] args)
-------------	---------------------------------------

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Interfaz

Interfaz()

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Ejercicio 1 (30/1/2003) e/s, java.util

En este ejercicio se concatena los archivos de entrada que se proporcionan a través de la línea de comandos, creando un solo flujo del tipo **SequenceInputStream**, y sacando la información por pantalla. Una versión simplificada sería equivalente al comando **cat** de UNÍS. Sin embargo, en este ejemplo, si no se especifica ningún archivo los datos se leerán de la entrada estándar. La lectura se realizará carácter a carácter.

La clase **SequenceInputStream** del paquete **java.io** crea un único flujo (o *stream*) de entrada leyendo uno o más flujos de entrada de bytes. Se lee del primer flujo hasta que la entrada finaliza, seguidamente se lee del siguiente, y así sucesivamente hasta el último. Esta clase tiene dos constructores, uno que concatena dos flujos y el segundo para concatenar varios flujos:

```
SequenceInputStream(InputStream s1, InputStream s2)
```

```
SequenceInputStream(Enumeration e)
```

Enumeration es una interfaz que proporciona “una interacción ordenada” a una lista de objetos. **Enumeration** es análoga a **Iterator**, de hecho tiene dos métodos que, aunque tienen distinto nombre son equivalentes: **hasMoreElements** que se comporta como **hasNext** y **nextElement** que se comporta como **next**. Podemos obtener un objeto **Enumeration** a partir de un objeto colección(**Collection**) utilizando el método estático **enumeration** de la clase **Collections**:

```
Enumeration enumeration(Collection c).
```

ArrayList es una buena implementación de una lista donde sus elementos utilizan un array. Se propone crear un objeto del tipo **ArrayList** para crear una estructura con los flujos a concatenar. Se creará un **ArrayList** lo suficientemente grande como para almacenar tantos objetos **BufferedInputStream** como parámetros haya en la línea de comandos, estos envolverán a los flujos **FileInputStream** (**FileInputStream(String nomArchivo)**).

Constructores de ArrayList
ArrayList()
ArrayList(int CapInicial)
ArrayList(Collection col)

El método add (public boolean add(Object e)) añade un elemento a la colección

Las clases **Buffered** almacenan los datos en *buffers* para evitar que sucesivas operaciones **read** y **write** actúen directamente sobre el flujo. Un flujo de este tipo crea y salva sus argumentos en un flujo para uso posterior, pudiendo especificar (de forma opcional según el constructor utilizado) el tamaño del buffer. Todos los flujos **Buffered** admiten dos constructores:

```
BufferedInputStream(InputStream in)
```

```
BufferedInputStream(InputStream in, int size)
```

Implemente en Java el programa completo tal y como se ha especificado en este enunciado, en una clase donde todo el código esté en el método **main**. (Abriendo los ficheros cuyos nombres se indican en la línea de comandos, construya la “envolvente” buferada, construya la lista, y añada los elementos buferados a la lista. Obtenga una referencia del tipo **Enumeration** para obtener el flujo deseado. Cierre todos los flujos y realice el tratamiento de excepciones si fuera oportuno).

Solución del Ejercicio 1

```
/*
 * Feb03_1.java
 *
 * Solución al ejercicio 1 del examen de Febrero de 2003 de
 * Fundamentos de Telemática.
 */

import java.io.*;
import java.util.*;

public class Feb03_1 {
    public static void main(String args[]) throws IOException{
        InputStream in = null;

        if(args.length == 0){
            in = System.in;
        }else{
            FileInputStream tabla0[] = new FileInputStream[args.length];
            BufferedInputStream tabla1[] = new
BufferedInputStream[args.length];

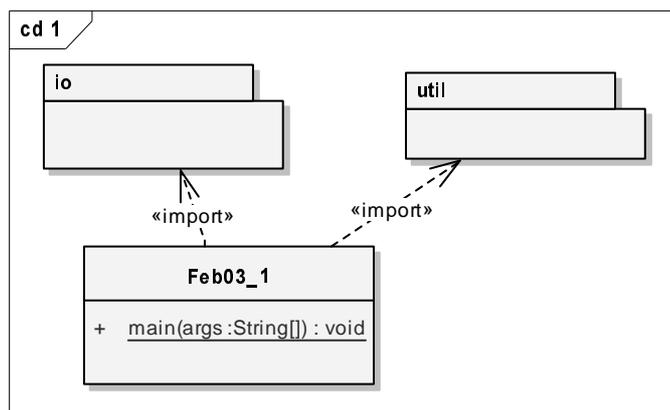
            List entradas = new ArrayList(args.length);
            for(int i=0; i<args.length; i++){
                tabla0[i] = new FileInputStream(args[i]);
                tabla1[i] = new BufferedInputStream(tabla0[i]);
                entradas.add(tabla1[i]);
            }

            Enumeration archivos = Collections.enumeration(entradas);
            in = new SequenceInputStream(archivos);

            int cr;

            while((cr = in.read()) != -1)
                System.out.println((char)cr);

            for(int i=0;i<args.length; i++){
                tabla0[i].close();
                tabla1[i].close();
            }
        }
    }
}
```



febrero03_1

Class Feb03_1

java.lang.Object

└─ febrero03_1.Feb03_1

```
public class Feb03_1
    extends java.lang.Object
```

Solución al Ejercicio 1 del examen de Febrero de 2003 de Fundamentos de Telemática.

Constructor Summary

Feb03_1()	
---------------------------	--

Method Summary

static void	main (java.lang.String[] args)
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Feb03_1

```
public Feb03_1()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
    throws java.io.IOException
```

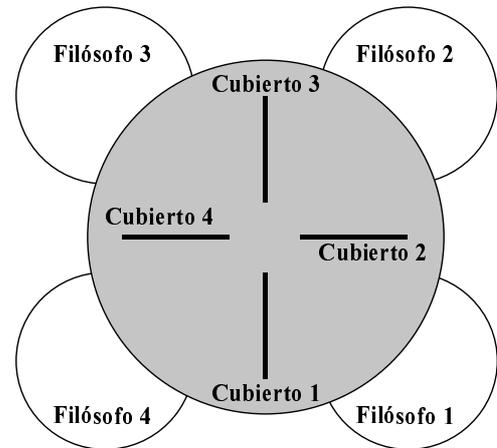
Throws:

java.io.IOException

Ejercicio 2 (30/1/2003) hilos, excepciones

Este ejercicio trata sobre sincronización: una serie de individuos (filósofos) sentados en una mesa redonda realizan dos funciones repetitivas: pensar y comer. Para pensar no necesitan ningún recurso, para comer necesitan dos cubiertos, los situados a derecha e izquierda de cada uno de los filósofos y que son recursos compartidos según la disposición del dibujo.

Se proporciona un programa Java con tres clases: **Cubiertos**, **Filosofo** y **Feb03_2**. La clase **Cubiertos** tiene una matriz que representa los cubiertos y con dos métodos **Coge**, y **Deja**, que realiza la operación de coger y dejar los cubiertos.



La clase **Filosofo** contiene la especificación de lo que realiza cada uno de los hilos. Y la clase **Feb03_2** que realiza la instanciación de los otros objetos.

```

/*****/
class Cubiertos {
    private boolean[] Cub=new boolean[4];
    public Cubiertos (){
        for(int i = 0; i<4; i++){
            Cub[i] = false;
        }
    }
    public void Coge(int i){
        Cub[i] = true;
        Cub[(i+1)%4] = true;
    }
    public void Deja(int i){
        Cub[i] = false;
        Cub[(i+1)%4] = false;
    }
}
/*****/
class Feb03_2{
    public static void main(String args[]){

        Cubiertos Cub = new Cubiertos();

        for(int i = 0; i<4; i++){
            new Filosofo(i,Cub)
        }
    }
}

```

```

/*****/
class Filosofo extends Thread{
    private int identi;
    private Cubiertos Cub;

    Filosofo (int i,, Cubiertos Cub){
        identi = i;
        this.Cub = Cub;
    }

    public void run(){
        while(true){
            pensar();
            Cub.Coge(identi);
            comer();
            Cub.Deja(identi);
        }
    }

    private void pensar(){}
    private void comer(){}
}

```

Modifique el código anterior (rescribiendo solamente lo que modifique):

1. Para que comiencen a ejecutarse los hilos.
2. Para que las operaciones estén sincronizadas de tal forma que, si cada filósofo en el el intento de hacerse con los dos cubiertos, tomara sólo uno de ellos (y no los dos cubiertos), el hilo se bloquea. Es decir, para que dos hilos (asociados a los filósofos) no puedan acceder a la vez a un recurso común (cubiertos), según la disposición del dibujo.

```
public final void wait() throws InterruptedException
```

Provoca que el hilo actual espere hasta que otro hilo invoque al método **notify()** o **notifyAll()**.

```
public final void notify()
```

Despierta el hilo que está esperando en el monitor del objeto.

```
void notifyAll()
```

Despierta todos los hilos que están esperando en el monitor del objeto.

Solución del Ejercicio 2

```
/*
 * Feb03_2.java
 *
 * Solución al Ejercicio 2 del examen de Febrero de 2003 de
 * Fundamentos de Telemática.
 */

public class Feb03_2{
    public static void main(String args[]){

        Cubiertos Cub = new Cubiertos();

        for(int i = 0; i<4; i++){
            new Filosofo(i,Cub);
        }
    }
}

/**
 * Clase Cubiertos con métodos sincronizados.
 */
class Cubiertos {
    private boolean[] Cub = new boolean[4];

    public Cubiertos (){
        for(int i = 0; i<4; i++){
            Cub[i] = false;
        }
    }

    public synchronized void Coge(int i){
        while(Cub[i] || Cub[(i+1)%4])
            try{
                System.out.println("Fil. "+i+" Espera");
                wait();
                System.out.print("hilo ");
                System.out.println(Thread.currentThread());
            }catch(InterruptedException e){}

        Cub[i] = true;
        Cub[(i+1)%4] = true;

        System.out.println("Fil. "+i+" Coge");
    }

    public synchronized void Deja(int i){
        Cub[i] = false;
        Cub[(i+1)%4] = false;
        notifyAll();
        System.out.println("Fil. "+i+" Deja");
    }
}
```

```

/**
 * Clase Filofo que hereda de Thread.
 */
class Filofo extends Thread{
    private int identi;
    private Cubiertos Cub;

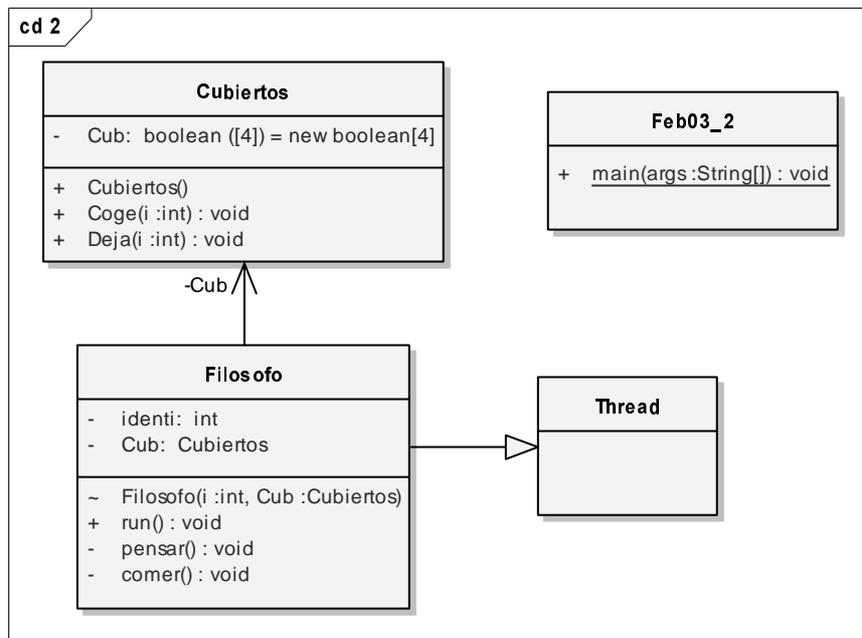
    Filofo (int i, Cubiertos Cub){
        identi = i;
        this.Cub = Cub;
    }

    public void run(){
        while(true){
            pensar();
            Cub.Coge(identi);
            comer();
            Cub.Deja(identi);
        }
    }

    private void pensar(){
        System.out.println("Fil. "+identi+" piensa");
        System.out.println(" "+identi+" piensa");
    }

    private void comer(){
        System.out.println("Fil. "+identi+" come");
        System.out.println(" "+identi+" come");
    }
}

```



febrero03_2

Class Feb03_2

java.lang.Object

└─ febrero03_2.Feb03_2

```
public class Feb03_2
    extends java.lang.Object
```

Solución al Ejercicio 2 del examen de Febrero de 2003 de Fundamentos de Telemática.

Constructor Summary

Feb03_2()	
---------------------------	--

Method Summary

static void	main (java.lang.String[] args)
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Feb03_2

```
public Feb03_2()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Ejercicio 3 (30/1/2003 y 10/7/2003) applet, awt

En este ejercicio se implementa un applet en la que se dibuja un ojo en el centro de la ventana y la pupila que se desplaza según la posición del ratón cuando está dentro de la ventana.

Se utilizará la técnica del *double buffer* para construir la imagen fuera de pantalla.

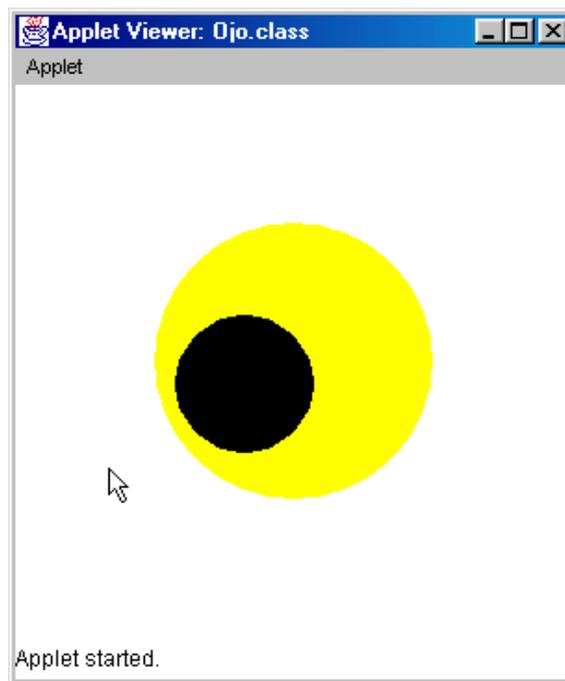
El fondo del applet será de color blanco, y el ojo se representará mediante dos círculos:

- El primero de ellos de ancho y alto la mitad de la anchura de las dimensiones de la ventana del applet, de color amarillo. Será una imagen fija.
- El segundo (pupila) de tamaño la mitad del círculo de color amarillo, será color negro y se desplazará en función de la posición del ratón, dentro (aprox.) del círculo de color amarillo, y que se desplace de forma gradual (lineal) en función de las coordenadas del ratón dentro de la ventana, usando la siguiente fórmula

✓ $x_{Actual} = (x_{Pupila} * 21 + x * 8) / 32$, para la posición x del ratón en la ventana.

✓ $y_{Actual} = (y_{Pupila} * 21 + y * 8) / 32$, para la posición y del ratón en la ventana.

donde x_{Pupila} e y_{Pupila} son los puntos de la pupila cuando se dibuja en el centro del ojo.



Se pide:

1. Etiqueta HTML para poder ejecutar el applet, si se desea una anchura y altura de 300 píxeles.
2. Importación de paquetes, nombre la clase e implemente el método `init()`, de tal forma que obtenga el objeto `Image` y obtenga el contexto gráfico a partir de la imagen.
3. Implementar el método `update()` de tal forma que llame a los métodos pertenecientes al applet que sean necesario y dibuje la imagen construida.
4. Implementar el método `paint()` de tal forma que sea donde se dibuje la imagen fuera de pantalla, configure el color de fondo, y dibuje el ojo (círculo interior y exterior) en la imagen.
5. Implemente el método para capturar el evento de movimiento del ratón, de tal forma que actualice la posición en la que se debe de colocar la pupila, vuelva a dibujar la imagen y realice un tratamiento del evento de tal forma que no siga buscando en el árbol de componentes.
6. ¿Se necesita, según lo especificado, algún método más? En caso afirmativo implementelo.

Algunos métodos contenidos en el paquete `java.awt`:

<code>Rectangle getBounds()</code>	Este método devuelve un objeto de la clase Rectangle . La clase Rectangle encapsula en un objeto las componentes (variables de instancia) width y height como enteros, con este método se puede obtener las dimensiones de la ventana.
<code>void setColor(Color c)</code>	Configurar el color del gráfico que se va a dibujar a continuación.
<code>Color getColor()</code>	Obtiene un color de las propiedades del sistema.
<code>Color getBackground()</code>	Obtiene el color de fondo.
<code>void fillOval(int x, int y, int ancho, int alto)</code>	Para dibujar una elipse rellena se utiliza el método fillOval() . La elipse se dibuja dentro del rectángulo cuya esquina superior izquierda corresponde a la posición (x,y) y cuyas dimensiones son las especificadas por ancho y alto . Para dibujar un círculo basta con que las dimensiones especificadas correspondan a las de un cuadrado en lugar de a las de un rectángulo.
<code>void fillRect(int x, int y, int width, int height)</code>	Llena el rectángulo. La forma de rellenar el rectángulo es análoga a como lo hace el método fillOval() .
<code>boolean drawImage(Image img, int x, int y, ImageObserver observer)</code>	Es un método sobrecargado y sólo se da información de la versión útil para este ejercicio. Dibuja la imagen tal y como esté disponible en ese momento. “ x ” e “ y ” deben ser 0 en el ejercicio ImageObserver debe ser null
<code>Graphics getGraphics()</code>	A partir de un objeto Image crea un contexto gráfico para dibujar una imagen fuera de pantalla
<code>Image createImage(int anchoApplet, int altoApplet)</code>	Crea una imagen fuera de pantalla para usarla en el doble buffer.

Para configurar el color amarillo, `static Color yellow`.

Para configurar el color blanco, `static Color white`.

Para configurar el color negro, `static Color black`.

Nota: Métodos de interés.

Los métodos de las applet	Eventos relacionados con el ratón
<code>public void init()</code> <code>public void start()</code> <code>public void destroy()</code> <code>public void paint(Graphics g)</code> <code>public void update(Graphics g)</code> <code>void repaint()</code>	<code>public boolean mouseMove(Event e, int x, int y)</code>

Solución del Ejercicio 3

```
/*
 * Applet Ojo.java
 *
 * Solución al Ejercicio 3 del examen de Febrero de 2003 de
 * Fundamentos de Telemática.
 *
 * <APPLET CODE="Ojo.class" WIDTH="300" HEIGHT="300">
 * <PARAM NAME="Cadena" VALUE="Esto sí que es chulo">
 * </APPLET>
 */

import java.applet.*;
import java.awt.*;

/**
 * Solución al Ejercicio 3 del examen de Febrero de 2003 de
 * Fundamentos de Telemática.
 *
 * @author Antonio Sierra
 */
public class Ojo extends Applet{
    int ancho, alto, anchoOjo, altoOjo, anchoPupila, altoPupila;
    int xOjo, yOjo, xPupila, yPupila, xActual, yActual;

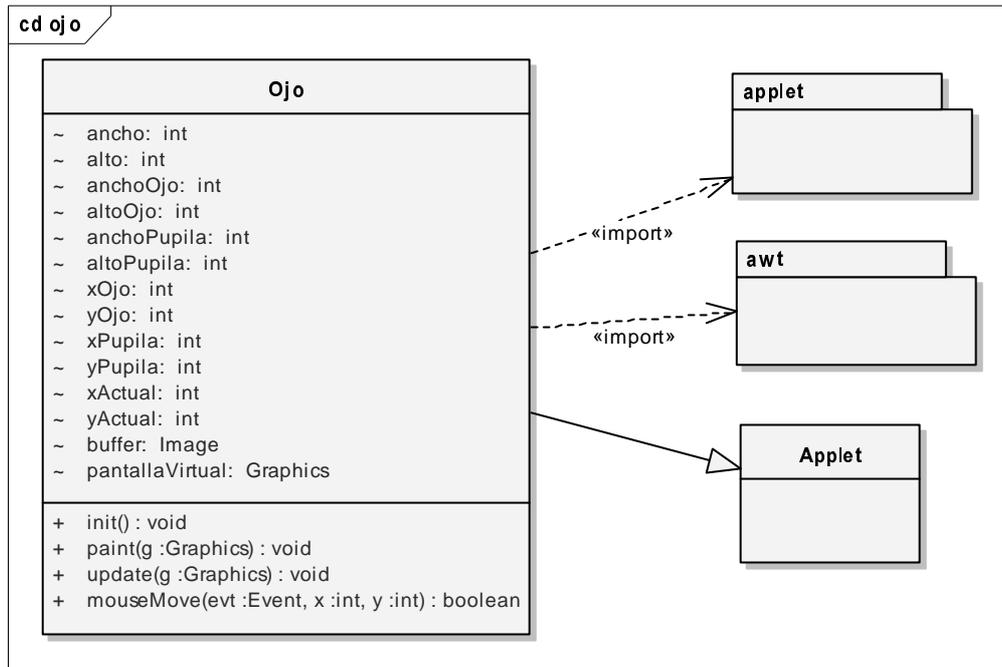
    Image buffer;
    Graphics pantallaVirtual;

    public void init(){
        ancho          = getBounds().width;
        alto           = getBounds().height;
        anchoOjo       = ancho/2;
        altoOjo        = alto/2;
        anchoPupila    = anchoOjo/2;
        altoPupila     = altoOjo/2;
        xOjo           = anchoOjo - anchoPupila;
        yOjo           = altoOjo - altoPupila;
        xPupila        = xOjo + anchoPupila/2;
        yPupila        = yOjo + altoPupila/2;
        xActual        = xPupila;
        yActual        = yPupila;
        buffer         = createImage(ancho,alto);
        pantallaVirtual = buffer.getGraphics();
    }

    public void paint(Graphics g){
        g.setColor(Color.white);
        g.fillRect(0, 0, ancho, alto);
        g.setColor(Color.yellow);
        g.fillOval(xOjo, yOjo, anchoOjo, altoOjo);
        g.setColor(Color.black);
        g.fillOval(xActual, yActual, anchoPupila, altoPupila);
    }

    public void update(Graphics g){
        Color colorTemporal = pantallaVirtual.getColor();
        pantallaVirtual.setColor(Color.white);
        pantallaVirtual.fillRect(0, 0, ancho, alto);
        pantallaVirtual.setColor(colorTemporal);
        paint(pantallaVirtual);
        g.drawImage(buffer, 0, 0, null);
    }
}
```

```
}  
    }  
    public boolean mouseMove(Event evt, int x, int y){  
        xActual = (xPupila*21 + x*8)/32;  
        yActual = (yPupila*21 + y*8)/32;  
        repaint();  
        return true;  
    }  
}
```



febrero03_3

Class Ojo

```
java.lang.Object
├── java.awt.Component
│   ├── java.awt.Container
│   │   ├── java.awt.Panel
│   │   │   ├── java.applet.Applet
│   │   │   └── febrero03_3.Ojo
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable,
javax.accessibility.Accessible

```
public class Ojo
    extends java.applet.Applet
```

Solución al Ejercicio 3 del examen de Febrero de 2003 de Fundamentos de Telemática.

See Also:

[Serialized Form](#)

Nested Class Summary

Nested classes/interfaces inherited from class java.applet.Applet

java.applet.Applet.AccessibleApplet

Nested classes/interfaces inherited from class java.awt.Panel

java.awt.Panel.AccessibleAWTPanel

Nested classes/interfaces inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

Nested classes/interfaces inherited from class java.awt.Component

java.awt.Component.AccessibleAWTComponent,
java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Ojo()

Method Summary

void init()

boolean mouseMove(java.awt.Event evt, int x, int y)

void paint(java.awt.Graphics g)

void update(java.awt.Graphics g)

Methods inherited from class java.applet.Applet

destroy, getAccessibleContext, getAppletContext, getAppletInfo, getAudioClip, getAudioClip, getCodeBase, getDocumentBase, getImage, getImage, getLocale, getParameter, getParameterInfo, isActive, newAudioClip, play, play, resize, resize, setStub, showStatus, start, stop

Methods inherited from class java.awt.Panel

addNotify

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Ojo

```
public Ojo()
```

Method Detail

init

```
public void init()
```

Overrides:

init in class java.applet.Applet

paint

```
public void paint(java.awt.Graphics g)
```

Overrides:

paint in class java.awt.Container

update

```
public void update(java.awt.Graphics g)
```

Overrides:

update in class java.awt.Container

mouseMove

```
public boolean mouseMove(java.awt.Event evt,  
                           int x,  
                           int y)
```

Overrides:

mouseMove in class java.awt.Component

[Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Ejercicio 1 (10/7/2003) applet, awt, hilos.

Se desea implementar un applet utilizando la técnica del *dobles almacenamiento temporal de gráficos* (o *dobles buffer de gráficos*), donde se usarán dos imágenes que se irán mostrando alternativamente cada segundo (haciendo uso de un hilo), en la pantalla del applet. Para utilizar el doble buffer, primero se crea un objeto **Image** en blanco, se dibuja en este objeto (usando métodos de la clase **Graphics**) y después se exhibe la imagen.

El método **getImage()**, devuelve un objeto **Image** y lo usaremos para cargar una imagen desde un fichero, en este ejercicio de la siguiente forma para obtener la imagen del fichero **imagen0.gif** e **imagen1.gif** del directorio **images** será (con **i** un valor de **0** o **1**):

```
getImage(getDocumentBase(), "images/imagen"+i+".gif");
```

1. Ponga la etiqueta HTML necesaria si las dimensiones son 500 y 400 píxeles.
2. Implemente el método **init()**, que obtenga un contexto gráfico para la imagen fuera de pantalla. Use una tabla de objetos **Image** para cargar las imágenes y haga que comience en este método la ejecución del hilo.
3. Implemente el método **run()**, que en un bucle infinito realice las llamadas necesarias para que otro(s) método(s) vuelquen las imágenes en la pantalla.
4. Implemente el método más oportuno (**paint()**, **update()** o ambos) para sacar las imágenes por pantalla.
5. ¿Es necesario implementar algún otro método del applet? En caso afirmativo deberá de codificarlo.

Solución Problema 1

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

public class Jun03_1 extends Applet implements Runnable{
    Thread hilo = null;
    private Image tabla[];
    private int totalImages = 2, //numero total de imagenes
               ImagenActual = 0, //Subindice de la actual
               sleepTime = 1000; //milisegundos que duerme
    //Para el doble buffer
    private Graphics gContext; //contexto gráfico fuera de pantalla
    private Image buffer; //buffer en el que se dibuja la imagen

    public void init(){
        tabla = new Image[totalImages];
        buffer = createImage(500,400); //crear buffer de imagen
        gContext = buffer.getGraphics(); //obtener contexto grafico

        for(int i = 0; i < tabla.length; i++){
            tabla[i]=getImage(getDocumentBase(),"images/imagen"+i+".gif");
            if(hilo == null){
                hilo = new Thread(this);
                hilo.start();
            }
        }

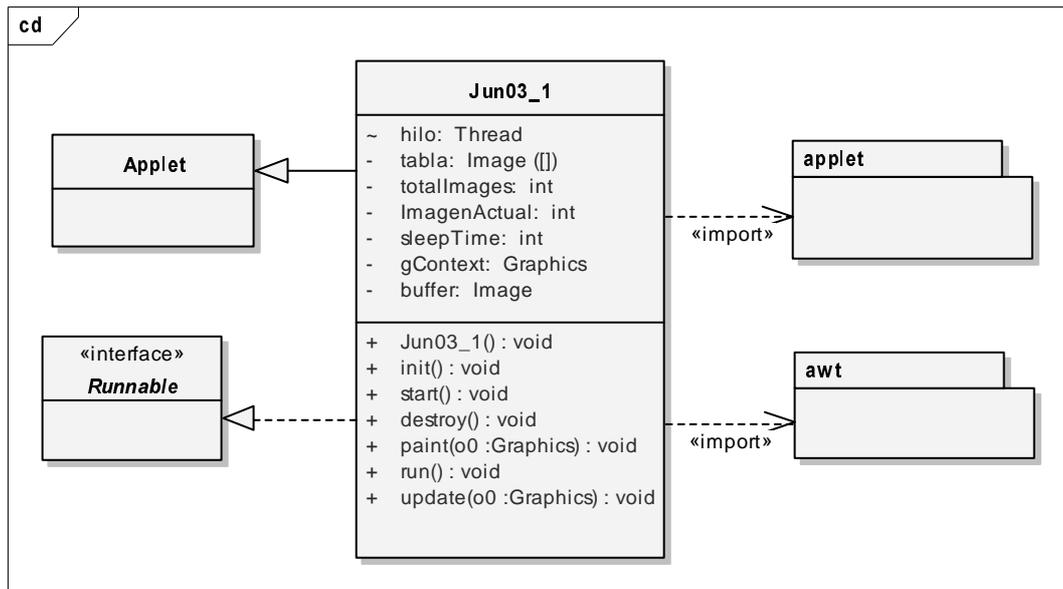
        //arranca la applet
        public void start(){
            if(hilo == null){
                hilo = new Thread(this);
                hilo.start();
            }
        }

        public void destroy(){
            if(hilo != null){
                hilo = null;
            }
        }

        //exhibir la imagen en el contexto gráfico de la applet
        public void paint(Graphics g){
        }

        public void run(){
            while(true){
                repaint();
                try{
                    Thread.sleep(sleepTime);
                }catch(InterruptedException e){/*showStatus(e.toString());*/}
            }
        }

        public void update(Graphics g){
            gContext.fillRect(0,0,500,400);
            gContext.drawImage(tabla[ImagenActual],0,0,this);
            ImagenActual = ++ImagenActual%totalImages ;
            g.drawImage(buffer,0,0,this);
            showStatus("Valor de la Imagen actual"+ImagenActual);
        }
    }
}
```



Class Jun03_1

java.lang.Object

java.awt.Component

java.awt.Container

java.awt.Panel

java.applet.Applet

Jun03_1

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, java.lang.Runnable, java.io.Serializable

```
public class Jun03_1
extends java.applet.Applet
implements java.lang.Runnable
```

See Also:

[Serialized Form](#)

Nested Class Summary

Nested classes inherited from class java.applet.Applet

java.applet.Applet.AccessibleApplet

Nested classes inherited from class java.awt.Panel

java.awt.Panel.AccessibleAWTPanel

Nested classes inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

Nested classes inherited from class java.awt.Component

java.awt.Component.AccessibleAWTComponent,
java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

private java.awt.Image	<u>buffer</u>
private java.awt.Graphics	<u>gContext</u>
(package private) java.lang.Thread	<u>hilo</u>
private int	<u>ImagenActual</u>
private int	<u>sleepTime</u>
private java.awt.Image[]	<u>tabla</u>
private int	<u>totalImages</u>

Fields inherited from class java.applet.Applet

Fields inherited from class java.awt.Panel

Fields inherited from class java.awt.Container

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT,
TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Jun03 1 ()

Method Summary

void destroy ()

void	<u>init</u> ()
void	<u>paint</u> (java.awt.Graphics g)
void	<u>run</u> ()
void	<u>start</u> ()
void	<u>update</u> (java.awt.Graphics g)

Methods inherited from class java.applet.Applet

getAccessibleContext, getAppletContext, getAppletInfo, getAudioClip, getAudioClip, getCodeBase, getDocumentBase, getImage, getImage, getLocale, getParameter, getParameterInfo, isActive, newAudioClip, play, play, resize, resize, setStub, showStatus, stop

Methods inherited from class java.awt.Panel

addNotify

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, addImpl, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getInsets, getLayout, getListeners, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, paramString, preferredSize, print, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, removeNotify, setFocusCycleRoot, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, setLayout, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getGraphicsConfiguration, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer.

```
getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize,
getToolkit, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent,
hasFocus, hide, imageUpdate, inside, isBackgroundSet, isCursorSet,
isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner,
isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque,
isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location,
lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp,
move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, printAll,
processComponentEvent, processFocusEvent, processHierarchyBoundsEvent,
processHierarchyEvent, processInputMethodEvent, processKeyEvent,
processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, remove,
removeComponentListener, removeFocusListener, removeHierarchyBoundsListener,
removeHierarchyListener, removeInputMethodListener, removeKeyListener,
removeMouseListener, removeMouseMotionListener, removeMouseWheelListener,
removePropertyChangeListener, removePropertyChangeListener, repaint, repaint,
requestFocus, requestFocusInWindow, requestFocus, requestFocusInWindow,
reshape, setBackground, setBounds, setBounds,
setComponentOrientation, setCursor, setDropTarget, setEnabled, setFocusable,
setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale,
setLocation, setLocation, setName, setSize, setSize, setVisible, show, show,
size, toString, transferFocus, transferFocusUpCycle
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait,
wait
```

Field Detail

hilo

```
java.lang.Thread hilo
```

tabla

```
private java.awt.Image[] tabla
```

totalImages

```
private int totalImages
```

ImagenActual

```
private int ImagenActual
```

sleepTime

```
private int sleepTime
```

gContext

```
private java.awt.Graphics gContext
```

buffer

```
private java.awt.Image buffer
```

Constructor Detail

Jun03_1

```
public Jun03_1()
```

Method Detail

init

```
public void init()
```

start

```
public void start()
```

destroy

```
public void destroy()
```

paint

```
public void paint(java.awt.Graphics g)
```

run

```
public void run()
```

Specified by:

run in interface java.lang.Runnable

update

```
public void update(java.awt.Graphics g)
```

Ejercicio 2 (10/7/2003) java.util,

Las clases contenedoras son potentes herramientas para el programador. Los contenedores de Java 2 son un rediseño de las clases que se incorporaban en Java 1.0 y 1.1. Los contenedores de Java 2 realizan la tarea de “almacenar objetos” y lo divide en dos conceptos diferentes:

1. Colección (**Collection**): grupo de elementos individuales, a los que generalmente se aplica alguna regla. (Una lista, **List**, debe contener elementos en una secuencia concreta, y un conjunto, **Set**, no puede tener duplicados).
2. Mapa (**Map**): grupo de pares de objetos *clave-valor*.

La visualización de los elementos de un contenedor puede realizarse con el método **System.out.println**, imprimiéndose por pantalla todos los elementos separados por una coma. En una Colección se imprime entre corchetes ([]) y un Mapa se imprime entre llaves ({ }), y si el mapa es **HashMap**, los valores separados por comas son las parejas **clave=valor**.

Para añadir elementos a una Colección se puede utilizar el método **add** (**public boolean add(Object e)**). Para añadir elementos a un Mapa se puede utilizar el método **put** (**Object put(Object clave, Object valor)**).

- Implemente todo el programa en una(s) clase(s) para que la llamada las llamadas desde **main** produzcan las salidas que se especifican en la siguiente tabla:

Llamada desde <i>main</i>	Salida
<code>System.out.println(rellenar(new ArrayList()));</code> <code>System.out.println(rellenar(new HashMap()));</code>	[Perro, Perro, Gato] {Gato=Klen, Perro=Platly}

- Escriba el código que falta a la siguiente tabla para que se produzca el resultado que se especifica.

Programa incompleto	Resultado por pantalla
<pre>import java.util.*; class ColeccionJun03_2b{ public static void main(String [] args){ ArrayList Gatos = new ArrayList(); for(int i = 0; i < 7; i++) Gatos.add(new Gato(i)); for(int i = 0; i < Gatos.size(); i++) ((Gato)Gatos.get(i)).escribir(); } }</pre>	<p>Gato #0 Gato #1 Gato #2 Gato #3 Gato #4 Gato #5 Gato #6</p>

El constructor sin parámetros de **HashMap**, da un tamaño inicial de 16 y el de **ArrayList**, de 10.

Algunos constructores de HashMap	Constructores de ArrayList
HashMap ()	ArrayList ()
HashMap (int CapInicial)	ArrayList(int CapInicial)
HashMap (Map m)	ArrayList(Collection col)
HashMap(int CapInicial, float loadFactor)	

Se extraen elementos de **ArrayList** método **get** . Podemos ver cuantos elementos tiene el **ArrayList** con **size ()**.

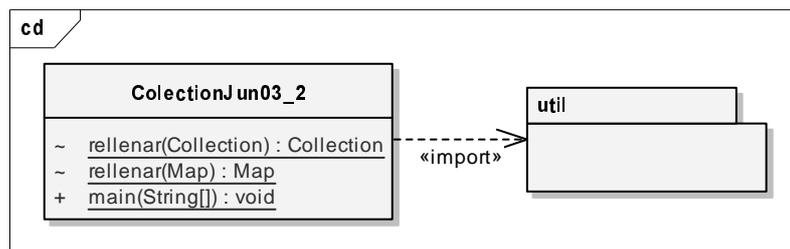
Ejercicio 2: Solución

Apartado 1

```
import java.util.*;

class ColeccionJun03_2{
    static Collection rellenar(Collection c){
        c.add("Perro");
        c.add("Perro");
        c.add("Gato");
        return c;
    }
    static Map rellenar(Map c){
        c.put("Perro", "Basiclay");
        c.put("Perro", "Platly");
        c.put("Gato", "Klen");
        return c;
    }

    public static void main(String [] args){
        System.out.println(rellenar(new ArrayList()));
        System.out.println(rellenar(new HashMap()));
    }
}
```



Class *ColectionJun03_2*

java.lang.Object

CollectionJun03_2

class **CollectionJun03_2**
extends java.lang.Object

Constructor Summary

(package private)	<u>CollectionJun03_2()</u>
-------------------	--

Method Summary

static void	<u>main</u> (java.lang.String[] args)
(package private) static java.util.Collection	<u>rellenar</u> (java.util.Collection c)
(package private) static java.util.Map	<u>rellenar</u> (java.util.Map c)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ColectionJun03_2

[CollectionJun03_2\(\)](#)

Method Detail

rellenar

static java.util.Collection **rellenar**(java.util.Collection c)

rellenar

static java.util.Map **rellenar**(java.util.Map c)

main

public static void **main**(java.lang.String[] args)

Apartado 2

```
import java.util.*;

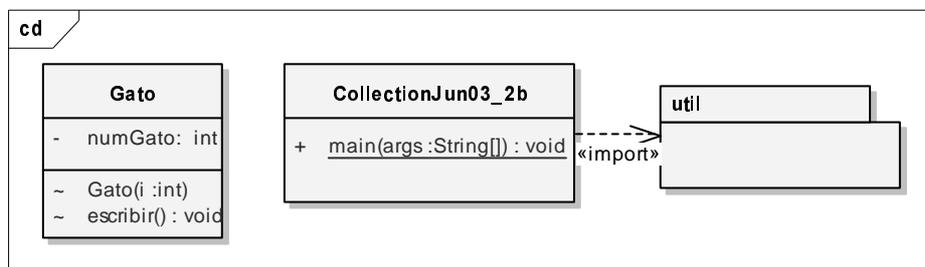
class CollectionJun03_2b{

    public static void main(String [] args){
        ArrayList Gatos = new ArrayList();

        for(int i = 0; i < 7; i++)
            Gatos.add(new Gato(i));

        for(int i = 0; i<Gatos.size(); i++)
            ((Gato)Gatos.get(i)).escribir();
    }
}

class Gato{
    private int numGato;
    Gato(int i){
        numGato = i;
    }
    void escribir(){
        System.out.println("Gato #" + numGato);
    }
}
```



Class CollectionJun03_2b

java.lang.Object

CollectionJun03_2b

class **CollectionJun03_2b**
extends java.lang.Object

Constructor Summary

(package private)	CollectionJun03_2b()
-------------------	--------------------------------------

Method Summary

static void	main (java.lang.String[] args)
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CollectionJun03_2b

CollectionJun03_2b()

Method Detail

main

public static void **main**(java.lang.String[] args)

Ejercicio 1 (15/9/2003) e/s, interfaz, excepciones

En este ejercicio se crea un objeto, se guarda en un fichero, se recupera del fichero y se muestra por pantalla. La *Serialización* de objetos da la posibilidad de salvar objetos con el *stream* de bytes (nunca de caracteres) en un archivo de disco o en una base de datos, y reconstruirse (*Deserialización*) posteriormente en forma de “objeto vivo”. Los *stream Object* (`ObjectInputStream` y `ObjectOutputStream`) permiten leer/escribir (`writeObject/readObject`) los bytes que representan al objeto.

Cuando un `ObjectOutputStream` escribe un objeto serializado, este objeto debe implementar la interfaz `Serializable`. Esta interfaz declara que la clase está diseñada de tal forma que sus objetos puedan ser serializados. En la versión por defecto se pueden serializar todos los campos que no sean ni `transient` ni `static`.

Implemente TODO el código del programa en el lenguaje Java de tal forma que los objetos a serializar tengan un solo campo de tipo entero, se escriba el objeto en un fichero de nombre `obj.txt`, se recupere y se imprima por pantalla.

<i>Constructores de FileInputStream</i>
<code>FileInputStream (String archivo) throws FileNotFoundException</code>
<code>FileInputStream (FileDescriptor fdObj) throws FileNotFoundException</code>
<code>FileInputStream (File file) throws FileNotFoundException</code>
<i>Constructores de FileOutputStream</i>
<code>FileOutputStream (String nombre) throws IOException</code>
<code>FileOutputStream (File file) throws IOException</code>
<code>FileOutputStream (FileDescriptor fd) throws IOException</code>
<code>FileOutputStream (String nombre, boolean append) throws IOException</code>
<i>Constructor de ObjectInputStream</i>
<code>ObjectInputStream (InputStream out) throws IOException</code>
<i>Constructor de ObjectOutputStream</i>
<code>ObjectOutputStream (OutputStream out) throws IOException</code>

Tenga en cuenta la excepción `FileNotFoundException`.

<i>Método de ObjectInputStream</i>
<code>void writeObject(Object obj) throws IOException, ClassNotFoundException</code>
<i>Método de ObjectOutputStream</i>
<code>Object readObject() throws IOException</code>

Solución del Ejercicio 1

```
/*
 * Ejercicio1.java
 *
 * Solución al ejercicio 1 del examen de Septiembre de 2003 de
 * Fundamentos de Telemática.
 */

import java.io.*;

/**
 * La clase Nombre implementa la interfaz Serializable
 */
class Nombre implements java.io.Serializable{
    private int valor = 7;
    int presenta(){
        return valor;
    }
}

/**
 * Clase Ejercicio1 que contiene el método Main
 */
public class Ejercicio1{

    /**
     * @param args Argumentos de la línea de comandos
     */
    public static void main(String args[]){
        try{
            FileOutputStream archivoSalida = new FileOutputStream ("obj.txt");

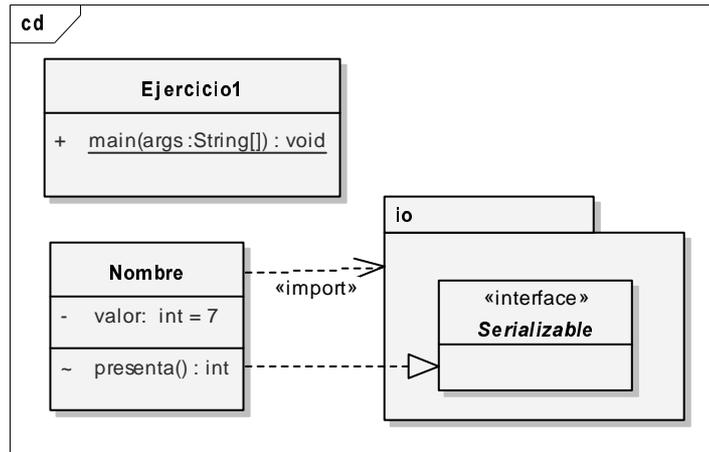
            ObjectOutputStream out = new ObjectOutputStream (archivoSalida);
            Nombre objeto = new Nombre();

            out.writeObject(objeto);

            archivoSalida.close();
            out.close();

            FileInputStream archivoEntrada = new FileInputStream ("obj.txt");
            ObjectInputStream in = new ObjectInputStream (archivoEntrada);
            Nombre obj = (Nombre) in.readObject();
            System.out.println(" "+obj.presenta());
            archivoEntrada.close();
            in.close();

        } catch (FileNotFoundException e){
            System.out.println("FileNotFoundException ");
        } catch (IOException e){
            System.out.println("IOException ");
        } catch (ClassNotFoundException e){
            System.out.println("ClassNotFoundException ");
        }
    }
}
```



Class Ejercicio1

java.lang.Object

Ejercicio1

```
public class Ejercicio1
extends java.lang.Object
```

Clase Ejercicio1 que contiene el método Main

Constructor Summary

Ejercicio1 ()

Method Summary

static void **main** (java.lang.String[] args)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ejercicio1

```
public Ejercicio1()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Parameters:

args - Argumentos de la línea de comandos

Ejercicio 2 (15/9/2003) java.util

En este ejercicio se utiliza un iterador creado a partir de un **ArrayList**, y los elementos se sacan por pantalla. Un iterador es un objeto para moverse a lo largo de una secuencia de elementos (objetos) y seleccionar cada uno de ellos sin que el programador (o usuario) tenga que saber la estructura subyacente de esta secuencia. Los iteradores se suelen llamar objetos “ligeros”, ya que es un objeto “fácil de crear” (por esto, suele encontrarse restricciones extrañas, como por ejemplo el que sólo se puedan mover en una dirección).

Los objetos **Iterator** son un ejemplo de iteradores con este tipo de limitaciones. Las operaciones que se pueden realizar son bastante restringidas. Se pueden realizar sólo las siguientes tareas:

1. Pedir a una colección que proporcione un iterador con el método **iterator()**. Este método pertenece a la interfaz **Collection**, y devuelve un objeto **Iterator** listo para devolver el primer elemento de la secuencia en la primera llamada a su método **next()** (**public Object next()**).
 2. Conseguir el siguiente objeto de la secuencia con **next()** (**public Object next()**).
 3. Comprobar si existen más objetos en la secuencia con **hasNext()** (**public boolean hasNext()**).
 4. Eliminar el último elemento devuelto por el iterador con **remove()** (**public void remove()**).
- Escriba el código que falta a la siguiente tabla para que se produzca el resultado que se especifica utilizando un iterador.

Programa incompleto	Resultado por pantalla
<pre>class Gato{ private int numGato; Gato(int i){ numGato = i; } void escribir(){ System.out.println("Gato #" + numGato); } }</pre>	Gato #0 Gato #1 Gato #2 Gato #3 Gato #4 Gato #5 Gato #6

Los constructores de **ArrayList** son:

```
ArrayList()
ArrayList(int Capinicial)
ArrayList(Collection col)
```

El constructor sin parámetros de **ArrayList** da un tamaño inicial de 10. Para añadir elementos a una *Colección* se puede utilizar el método **add** (**public boolean add(Object e)**).

Solución del Ejercicio 2

```
/*
 * Ejercicio2.java
 *
 * Solución al ejercicio 2 del examen de Septiembre de 2003 de
 * Fundamentos de Telemática.
 */

import java.util.*;

class Gato{
    private int numGato;
    Gato(int i){
        numGato = i;
    }
    void escribir(){
        System.out.println("Gato #" + numGato);
    }
}

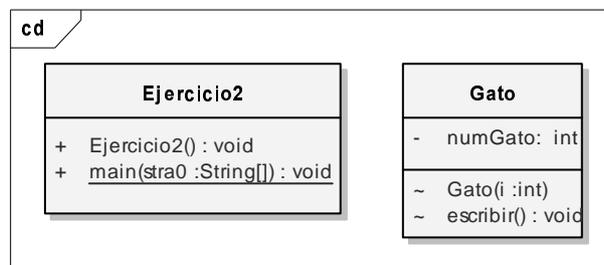
/**
 * Clase Ejercicio2 que contiene el método main
 */
public class Ejercicio2{

    /**
     * @param args Argumentos de la línea de comandos
     */
    public static void main(String args[]){
        ArrayList Gatos = new ArrayList();

        for(int i = 0; i < 7; i++)
            Gatos.add(new Gato(i));

        Iterator e = Gatos.iterator();

        while(e.hasNext())
            ((Gato)e.next()).escribir();
    }
}
```



Class Ejercicio2

java.lang.Object
└─ **Ejercicio2**

```
public class Ejercicio2  
    extends java.lang.Object
```

Solución al ejercicio 2 del examen de Septiembre de 2003 de Fundamentos de Telemática.

Constructor Summary

Ejercicio2()	
------------------------------	--

Method Summary

static void	main (java.lang.String[] args)
-------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ejercicio2

```
public Ejercicio2()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Parameters:

args - Argumentos de la línea de comandos

Ejercicio 3 (15/9/2003) java.util, excepciones

Las listas son objetos modificables, y en este ejercicio se implementará una subclase de **AbstractList** que proporcione una implementación redefiniendo el método **set**, para que lance la excepción **UnsupportedOperationException** si se tuviera que modificar (aumentar) el tamaño de memoria reservado para la lista en el momento de llamar al constructor.

Normalmente, siempre existe alguna implementación de colecciones que satisfaga las necesidades como programador. Si no es así, se pueden implementar las clases abstractas que se proporcionan (**AbstractCollection**, **AbstractSet**, **AbstractList**, **AbstractMap**, **AbstractSequentialList**).

Todas las clases abstractas de colecciones declaran algunos métodos abstractos. **AbstractList** tiene dos métodos abstractos: **size** y **get**. Todos los métodos de consulta de **AbstractList** se implementan utilizando estos métodos, incluyendo a sus iteradores. Sólo es necesario escribir nuestra propia implementación de los otros métodos. Normalmente se realiza para aumentar la eficiencia o para permitir algo que por defecto no está permitido.

1. (1p) Crear la clase **ListaMuchosArrays** que herede de **AbstractList** con dos variables de instancia privadas **arrays** (del tipo **Object[][]**) y **tam** (del tipo entero).
2. (3p) Constructor de la clase al que se le pasa como parámetro un objeto del tipo **Object[][]**, copia la tabla en la variable de instancia (guardándolo en una zona de memoria diferente) y determina el tamaño **tam**. (Puede hacer uso de **length**).
3. (1p) Implemente el método **size(public int size())** que devuelve el tamaño de la matriz.
4. (2p) Implemente el método **get(public Object get(int indice))** que devuelve el objeto cuyo índice se le pasa como parámetro. Si el valor del índice está fuera del rango de la tabla se deberá lanzar la excepción **ArrayIndexOutOfBoundsException**.
5. (3p) Implemente el método **set (public Object set(int indice, Object valor))** que asigne a la posición **indice** el objeto **valor**. Si el valor del índice está fuera del rango de la tabla se deberá lanzar la excepción **ArrayIndexOutOfBoundsException**.

Solución del Ejercicio 3

```
/*
 * ListaMuchosArrays.java
 *
 * Solución al ejercicio 3 del examen de Septiembre de 2003 de
 * Fundamentos de Telemática.
 */

import java.util.*;

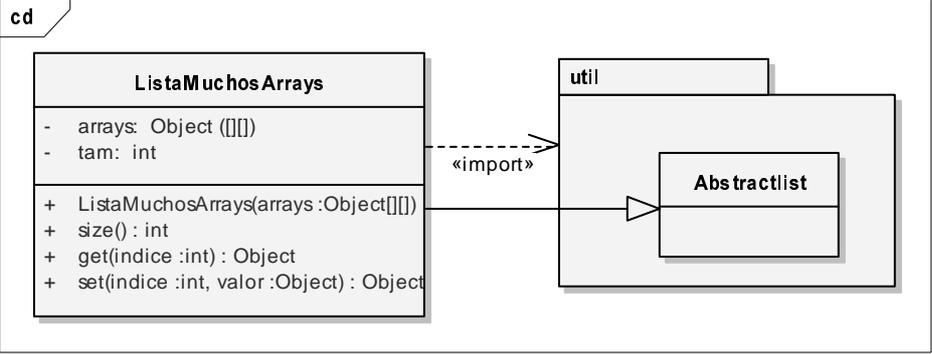
/**
 * Clase ListaMuchosArrays que hereda de AbstractList
 */
public class ListaMuchosArrays extends AbstractList{
    private Object[][] arrays;
    private int tamano;

    public ListaMuchosArrays(Object[][] arrays){
        this.arrays = (Object[][])arrays.clone();
        int s =0;
        for(int i = 0;i< arrays.length; i++)
            s += arrays[i].length;
        tamano = s;
    }

    /**
     * Método size que devuelve el tamaño de la matriz
     * @return tamano
     */
    public int size(){
        return tamano;
    }

    /**
     * Método get que devuelve el objeto cuyo índice se le pasa como
     * parámetro devuelve el tamaño de la matriz
     */
    public Object get (int indice){
        int desp=0; //desplazamiento desde el inicio de la colección
        for(int i = 0; i < arrays.length ; i++){
            if(indice < desp + arrays[i].length)
                return arrays[i][indice-desp];
            desp += arrays[i].length;
        }
        throw new ArrayIndexOutOfBoundsException(indice);
    }

    /**
     * Método set que asigna a la posición indice el objeto valor
     */
    public Object set (int indice, Object valor){
        int desp = 0; //desplazamiento desde el inicio de la colección
        for(int i = 0; i < arrays.length ; i++){
            if(indice < desp + arrays[i].length){
                Object ret = arrays[i][indice-desp];
                arrays[i][indice-desp] = valor;
                return ret;
            }
            desp += arrays[i].length;
        }
        throw new ArrayIndexOutOfBoundsException(indice);
    }
}
```



Class ListaMuchosArrays

```
java.lang.Object
├─ java.util.AbstractCollection<E>
│   └─ java.util.AbstractList
│       └─ ListaMuchosArrays
```

All Implemented Interfaces:

java.lang.Iterable, java.util.Collection, java.util.List

```
public class ListaMuchosArrays
extends java.util.AbstractList
```

Solución al ejercicio 3 del examen de Septiembre de 2003 de Fundamentos de Telemática.

Field Summary

Fields inherited from class java.util.AbstractList

modCount

Constructor Summary

ListaMuchosArrays(java.lang.Object[][] arrays)

Method Summary

java.lang.Object	get (int indice) Método get que devuelve el objeto cuyo índice se le pasa como parámetro devuelve el tamaño de la matriz
java.lang.Object	set (int indice, java.lang.Object valor) Método set que asigna a la posición indice el objeto valor
int	size () Método size que devuelve el tamaño de la matriz

Methods inherited from class java.util.AbstractList

add, add, addAll, clear, equals, hashCode, indexOf, iterator, lastIndexOf, listIterator, listIterator, remove, removeRange, subList

Methods inherited from class java.util.AbstractCollection

addAll, contains, containsAll, isEmpty, remove, removeAll, retainAll, toArray, toArray, toString

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.List

addAll, contains, containsAll, isEmpty, remove, removeAll, retainAll, toArray, toArray

Constructor Detail

ListaMuchosArrays

```
public ListaMuchosArrays(java.lang.Object[][] arrays)
```

Method Detail

size

```
public int size()
```

Método size que devuelve el tamaño de la matriz

Specified by:

size in interface java.util.Collection

Specified by:

size in interface java.util.List

Specified by:

size in class java.util.AbstractCollection

Returns:

tamano

get

```
public java.lang.Object get(int indice)
```

Método get que devuelve el objeto cuyo índice se le pasa como parámetro devuelve el tamaño de la matriz

Specified by:

get in interface java.util.List

Specified by:

get in class java.util.AbstractList

set

```
public java.lang.Object set(int indice,  
                             java.lang.Object valor)
```

Método set que asigna a la posición indice el objeto valor

Specified by:

set in interface java.util.List

Overrides:

set in class java.util.AbstractList

[Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

[DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

Ejercicio 1 (21/1/2004) java.util

En este ejercicio se implementa una pila y una cola a partir de un objeto `LinkedList`. A una pila se le puede denominar contenedor “*last-in, first-out*” o LIFO (el último en entrar es el primero en salir). Es decir, el último elemento en introducir en la pila será el primero que se extraiga. A una cola se le puede denominar contenedor “*first-in, first-out*” o FIFO (el primero en entrar es el primero en salir). Es decir, los elementos se extraerán en el mismo orden en que fueron introducidos.

- Implementar la clase `PilaL` con un objeto `LinkedList` y que implemente los métodos `apilar` que inserte un elemento a la estructura, `cima` que obtenga el elemento de la cima sin eliminarlo, y `desapilar` que extraiga un elemento. Diagrama de clases UML del código implementado (detallado).
- Implementar la clase `ColaL` con un objeto `LinkedList` y que implemente los métodos `poner` que inserte un elemento a la estructura, `estaVacía` que compruebe si una estructura no contiene ningún elemento, y `quitar` que extraiga un elemento. Diagrama de clases UML del código implementado (detallado).

(Nota: Especifique la visibilidad -privada, pública, ...- de los métodos, variables y clases).

Los constructores son `LinkedList()` y `LinkedList(Collection c)`.

Métodos de <code>LinkedList</code>	Explicación
<code>void add(int index, Object element)</code>	Inserta el objeto en la posición especificada.
<code>boolean add(Object o)</code>	Añade el elemento especificado al final.
<code>boolean addAll(Collection c)</code>	Añade los elementos al final.
<code>boolean addAll(int index, Collection c)</code>	Añade a partir del valor especificado
<code>void addFirst(Object o)</code>	Inserta al principio de la lista.
<code>void addLast(Object o)</code>	Añade al final de la lista.
<code>void clear()</code>	Borra todos los elementos
<code>Object clone()</code>	Devuelve una copia.
<code>boolean contains(Object o)</code>	Devuelve <code>true</code> si contiene un elemento.
<code>Object get(int index)</code>	Devuelve el elemento de la posición.
<code>Object getFirst()</code>	Devuelve el primer elemento.
<code>Object getLast()</code>	Devuelve el último elemento.
<code>int indexOf(Object o)</code>	Devuelve el índice de la primera ocurrencia (sino está -1)
<code>int lastIndexOf(Object o)</code>	Devuelve el índice de la última ocurrencia (sino está -1)
<code>ListIterator listIterator(int index)</code>	Construye un <code>ListIterator</code> a partir del índice indicado
<code>Object remove(int index)</code>	Borra el elemento con índice indicado.
<code>boolean remove(Object o)</code>	Borra la primera ocurrencia del elemento.
<code>Object removeFirst()</code>	Borra y devuelve el primer elemento.
<code>Object removeLast()</code>	Borra y devuelve el último elemento.
<code>Object set(int index, Object elem)</code>	Reemplaza el elemento de la posición.
<code>int size()</code>	Devuelve el número de elementos.
<code>Object[] toArray()</code>	Devuelve un array con todos los elementos
<code>Object[] toArray(Object[] a)</code>	Devuelve un array con todos los elementos

Solución al Ejercicio 1

```
/*
 * Principal.java
 *
 * Solución al Ejercicio 1 del examen de Febrero de 2004 de
 * Fundamentos de Telemática.
 */

import java.util.*;

public class Principal{

    /**
     * @param args Argumentos en la línea de comandos
     */
    public static void main(String args[]){
        PilaL pila = new PilaL();
        for (int i = 0; i<10; i++)
            pila.apilar(Integer.toString(i));
        System.out.println(pila.cima());
        System.out.println(pila.cima());
        System.out.println(pila.desapilar());
        System.out.println(pila.desapilar());
        System.out.println(pila.desapilar());

        ColaL cola = new ColaL();
        for (int i = 0; i<10; i++)
            cola.poner(Integer.toString(i));
        System.out.println(cola.quitar());
        System.out.println(cola.quitar());
    }

    /**
     * Clase PilaL con un objeto LinkedList que implementa los métodos
     * apilar, cima y desapilar.
     */
    class PilaL{
        private LinkedList lista = new LinkedList();

        public void apilar (Object v){
            lista.addFirst(v);
        }

        public Object cima(){
            return lista.getFirst();
        }
        public Object desapilar(){
            return lista.removeFirst();
        }
    }

    /**
     * Clase ColaL con un objeto LinkedList que implementa los métodos poner,
     * estaVacia y quitar.
     */
    class ColaL{
        private LinkedList lista = new LinkedList();

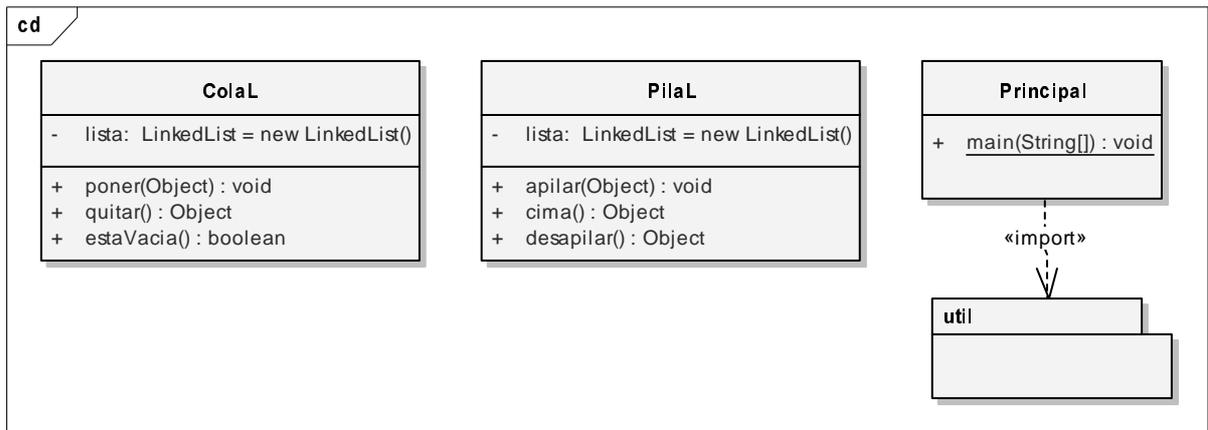
        public void poner (Object v){
```

```

        lista.addFirst(v);
    }

    public Object quitar(){
        return lista.removeLast();
    }
    public boolean estaVacia(){
        return lista.isEmpty();
    }
}

```



febrero04

Class Principal

java.lang.Object

└ febrero04.Principal

```
public class Principal
extends java.lang.Object
```

Solución al Ejercicio 1 del examen de Febrero de 2004 de Fundamentos de Telemática.

Constructor Summary

[Principal\(\)](#)

Method Summary

static void [main](#)(java.lang.String[] args)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Principal

```
public Principal()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Parameters:

args - Argumentos en la línea de comandos

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

Ejercicio 2 (21/1/2004) java.util

En este ejercicio se usa un objeto **HashMap**. Se va a relizar un programa que compruebe el correcto funcionamiento del método estático **random** de la clase **Math** del paquete **java.lang** (`static double random()`). De manera ideal, este método producirá una distribución perfecta (homogénea) de números entre el 0 y el 1. Para probarlo se generará un conjunto de número (10000 en total) y se contará cuántos caen en cada subrango del 0 al 1 (se “normalizará” estableciendo 20 subrangos discretos, usando objetos **Integer** que representen los valores del 0 al 19). Para realizar esta operación resulta adecuado utilizar un objeto **HashMap**, puesto que asocia objetos (número de ocurrencias del valor en un intervalo) con objetos (el valor “normalizado” del intervalo). El contenido del objeto **HashMap** debe sacarse por pantalla.

El objeto *valor* contendrá el número “normalizado” producido por `Math.random()`, junto con la cantidad de veces que aparece ese número que será la *clave*.

El constructor sin parámetros de **HashMap**, da un tamaño inicial de 16. El resto de constructores son `HashMap (int CapInicial)`, `HashMap (Map m)` y `HashMap(int CapInicial, float loadFactor)`.

Métodos de HashMap	Explicación
<code>void clear()</code>	Borra todos los elementos.
<code>Object clone()</code>	Devuelve una copia
<code>boolean containsKey(Object key)</code>	Devuelve <code>true</code> si el mapa contine la clave.
<code>boolean containsValue(Object value)</code>	Devuelve <code>true</code> si el mapa contine el valor.
<code>Set entrySet()</code>	Devuelve una vista de la colección que contiene este mapa.
<code>Object get(Object key)</code>	Devuelve el valor para el que se especifica la clave, o <code>null</code> si no lo contiene
<code>boolean isEmpty()</code>	Devuelve <code>true</code> si el mapa no contiene ningun mapeo clave-valor.
<code>Set keySet()</code>	Devuelve una vista <code>Set</code> que contiene las claves
<code>Object put(Object key, Object valor)</code>	Asocia un valor a una clave.
<code>void putAll(Map m)</code>	Copia todos las entradas de un mapa a otro.
<code>Object remove(Object key)</code>	Borra una entrada asociada con la clave.
<code>int size()</code>	Devuelve el número de entradas.
<code>Collection values()</code>	Devuelve una vista de los valores.

Los constructores de la clase **Integer** son `Integer(int value)` y `Integer(String s)`.

Solución al Ejercicio 2

```
/*
 * Principal.java
 *
 * Solución al Ejercicio 2 del examen de Febrero de 2004 de
 * Fundamentos de Telemática.
 */

import java.util.*;

public class Principal {

    /**
     * Cada vez que se genera un número aleatorio, se envuelve
     * en un objeto Integer de forma que pueda usarse la referencia con el
     * objeto HashMap. (No se puede usar una primitiva con un contenedor,
     * solo una referencia al objeto).
     *
     * @param args Argumentos en la línea de comandos
     */
    public static void main(String[] args) {
        HashMap hm = new HashMap();
        for(int i = 0; i < 10000; i++){
            Integer r = new Integer((int)(Math.random()*20));

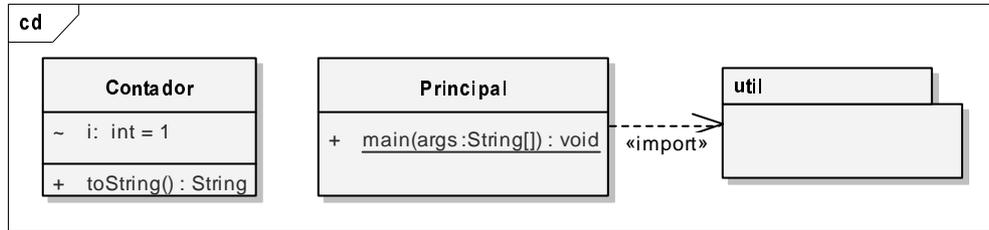
            /**
             *El método containsKey() comprueba si la clave ya está en el
             *contenedor.(Es decir, ¿se ha encontrado el número?) En caso
             *afirmativo, el método get() produce el valor asociado para
             *la clave, que en este caso es un objeto Contador.
             *El valor i del contenedor se incrementa para indicar que se ha
             *encontrado una ocurrencia más de este número al azar en
             *particular.
             */
            if(hm.containsKey(r))
                ((Contador)hm.get(r)).i++;

            /** Si no se ha encontrado aún la clave, el método put() ubicará
             *un nuevo par clave-valor en el objeto HashMap. Desde que el
             *Contador iniciacliza automáticamente su variable i a uno cuando
             *se crea, indica la primer ocurrencia de este número al azar en
             *concreto.
             */
            else
                hm.put(r,new Contador());
        }

        /**
         *Para mostrar el HashMap, simplemente se imprime. El método
         *toString() de HashMap se mueve por todos los pares clave-valor y
         *llama a toString() para cada uno. El Integer.toString() está
         *predefinido, y se puede ver el toString() para Contador.
         */
        System.out.println(hm);
    }
}
```

```
class Contador{
    int i = 1;

    public String toString(){
        return Integer.toString(i);
    }
}
```



febrero04_2

Class Principal

java.lang.Object

└ febrero04_2.Principal

```
public class Principal
extends java.lang.Object
```

Solución al Ejercicio 2 del examen de Febrero de 2004 de Fundamentos de Telemática.

Constructor Summary

[Principal\(\)](#)

Method Summary

static void [main](#)(java.lang.String[] args)

Cada vez que se genera un número aleatorio, se envuelve en un objeto Integer de forma que pueda usarse la referencia con el objeto HashMap.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Principal

```
public Principal()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Cada vez que se genera un número aleatorio, se envuelve en un objeto Integer de forma que pueda usarse la referencia con el objeto HashMap. (No se puede usar una primitiva con un contenedor, solo una referencia al objeto).

Parameters:

args - Argumentos en la línea de comandos

Ejercicio 1 (4/9/2004 y 14/2/2005)

interfaz, java.util

Una forma de establecer comparaciones con los objetos en Java es codificando clases que implementen la interfaz `java.lang.Comparable`, que establece una “*comparación natural*”. La interfaz `Comparable` contiene un solo método `compareTo (int compareTo (Object o))`, que compara el objeto con otro especificado para su ordenación. Este método toma otro objeto como parámetro, y produce un valor negativo si el parámetro es menor que el objeto actual, cero si el parámetro es igual, y un valor positivo si el parámetro es mayor que el objeto actual.

Se proporciona el siguiente código:

```
public class Conjunto2{
    public static Set rellenar(Set a, int tamaño){
        for(int i = 0; i < tamaño; i++){
            a.add(new MiTipo(i));
        }
        return a;
    }
    public static void prueba(Set a){
        rellenar(a, 10 );
        rellenar(a, 10 ); //intenta introducir duplicados
        rellenar(a, 10 );
        a.addAll(rellenar(new TreeSet(),10));
        System.out.println(a);
    }
}
class Ppal{
    public static void main(String args[]){
        Conjunto2.prueba(new HashSet());
        // Conjunto2.prueba(new TreeSet());
    }
}
```

- Se pide implementar TODA la clase **MiTipo**, coherente con el código arriba proporcionado, que realice la ordenación usando la interfaz **Comparable**, según el tipo entero proporcionado como parámetro al constructor. Sobreescriba los métodos **equals** (determinando cuando dos objetos son iguales), **hashCode** (que le asigne el entero del parámetro del constructor como código hash), **toString** (que devuelva como objeto **String** el valor del entero) coherente con código proporcionado arriba.
- ¿Qué salida produce la siguiente línea de código (línea de método **main**)?
`Conjunto2.prueba(new HashSet());`
- En el método **main**, hay una línea puesta como comentario,
`// Conjunto2.prueba(new TreeSet());`
Si se quitan las dos barras inclinadas, ¿qué salida produce?, ¿qué diferencia hay con la línea de arriba (`Conjunto2.prueba(new HashSet());`)?
- Diagrama de clases UML de todo el código. Hágalo de forma detallada.

Solución del Ejercicio 1

```
/*
 * Ppal.java
 *
 * Solución al ejercicio 1 del examen de Septiembre de 2004 de
 * Fundamentos de Telemática.
 */

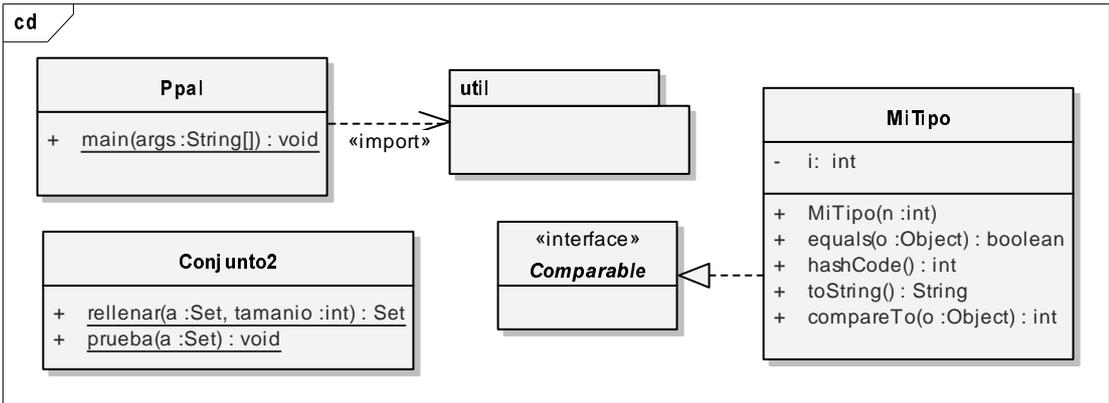
import java.util.*;

/**
 * La clase MiTipo implementa la interfaz Comparable
 */
class MiTipo implements Comparable{
    private int i;
    public MiTipo(int n){
        i = n;
    }
    public boolean equals(Object o){
        return (o instanceof MiTipo) && (i==((MiTipo)o).i);
    }
    public int hashCode(){
        return i;
    }
    public String toString(){
        return i + " ";
    }
    public int compareTo(Object o){
        int i2 = ((MiTipo)o).i;
        return (i2<i ? -1 : (i2 ==i ? 0 : 1));
    }
}

class Conjunto2{
    public static Set rellenar(Set a, int tamaño){
        for(int i = 0; i < tamaño; i++)
            a.add(new MiTipo(i));
        return a;
    }
    public static void prueba(Set a){
        rellenar(a, 10 );
        rellenar(a, 10 ); //intenta introducir duplicados
        rellenar(a, 10 );
        a.addAll(rellenar(new TreeSet(),10));
        System.out.println(a);
    }
}

/**
 * Clase Ppal que contiene el método Main
 */
public class Ppal{

    /**
     * @param args Argumentos de la línea de comandos
     */
    public static void main(String args[]){
        Conjunto2.prueba(new HashSet());
        Conjunto2.prueba(new TreeSet());
    }
}
```



Class Ppal

java.lang.Object

└ Ppal

```
public class Ppal
extends java.lang.Object
```

Solución al ejercicio 1 del examen de Septiembre de 2004 de Fundamentos de Telemática.

Constructor Summary

[Ppal\(\)](#)

Method Summary

static void [main](#)(java.lang.String[] args)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Ppal

```
public Ppal()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Parameters:

args - Argumentos de la línea de comandos

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

Ejercicio 2 (4/9/2004 y 14/2/2005)

java.util

En este ejercicio se trabaja con un objeto **HashMap**. Considere un sistema de predicción del tiempo que use objetos **Meteorologo** con objetos **Prediccion**. A cada **Meteorologo** se le da un número de identidad, de forma que se pueda buscar una **Prediccion** en el objeto **HashMap**.

```
import java.util.*;

class Meteorologo {
    int numMeteorologo;

    Meteorologo (int n){
        numMeteorologo = n;
    }
}

class Prediccion{
    boolean oscurecer = Math.random() > 0.5;
    public String toString(){
        if(oscurecer)
            return "Seis semanas más de invierno!!!";
        else
            return "primavera temprana";
    }
}

public class DetectorPrimavera{
    public static void main(String args[]){
        HashMap hm = new HashMap();
        for(int i = 0; i < 10; i++){
            hm.put(new Meteorologo(i),new Prediccion());
        }

        System.out.println("Buscando prediccion meteorologo #3");
        Meteorologo gh = new Meteorologo(3);

        if(hm.containsKey(gh))
            System.out.println((Prediccion)hm.get(gh));
        else
            System.out.println("Clave no encontrada: "+gh);
    }
}
```

Sin embargo, en el ejemplo al ejecutarlo, genera la siguiente salida:

```
Buscando prediccion meteorologo #3
Clave no encontrada
```

- Modifique (añada/quite) las clases **Meteorologo** y/o **Prediccion** para que el código del método **main**, obtenga la predicción del **Meteorologo (3)**. No modifique el código del método **main**. Use para ello haga uso de los métodos **equals (public boolean equals (Object key))** y **hashCode (public int hashCode ())**.
- Diagrama de clases UML de todo el código. Hágalo de forma detallada.

El constructor sin parámetros de **HashMap**, da un tamaño inicial de 16. El resto de constructores son **HashMap (int CapInicial)**, **HashMap (Map m)** y **HashMap(int CapInicial,float loadFactor)**.

Métodos de HashMap	
void clear()	Borra todos los elementos.
Object clone()	Devuelve una copia
boolean containsKey(Object key)	Devuelve true si el mapa contiene la clave.
Boolean containsValue(Object value)	Devuelve true si el mapa contiene el valor.
Set entrySet()	Devuelve una vista de la colección que contiene este mapa.
Object get(Object key)	Devuelve el valor para el que se especifica la clave, o null si no lo contiene
boolean isEmpty()	Devuelve true si el mapa no contiene ningún mapeo clave-valor.
Set keySet()	Devuelve una vista Set que contiene las claves
Object put (Object key, Object valor)	Asocia un valor a una clave.
void putAll(Map m)	Copia todos las entradas de un mapa a otro.
Object remove (Object key)	Borra una entrada asociada con la clave.
int size()	Devuelve el número de entradas.
Collection values()	Devuelve una vista de los valores.

Métodos de **Object**: **public boolean equals(Object key)** y **public int hashCode()** y **public String toString()**.

La expresión **(x instanceof Q)** devuelve cierto (**true**) si la referencia **x** es instancia de la clase **Q**.

Algunos métodos de la interfaz **Set**:

boolean add(Object o)	Añade el elemento al conjunto si no está ya presente.
boolean addAll(Collection o)	Añade todos los elementos al conjunto si no está ya presente.

Solución del Ejercicio 2

```
/*
 * El problema es que Meteorologo se hereda de Object. El método
 * hashCode() de Object el que se usa para generar el código hash
 * de cada objeto, y por defecto, usa únicamente la dirección del objeto.
 * Por tanto, la primera instancia de Meteorologo(3) NO produce un código
 * de hash igual al código de hash de la segunda instancia de
 * Meteorologo(3) que se intentó usar en la búsqueda.
 *
 * Hay además que hacer lo siguiente:
 * sobrescribir el método equals() que también es de Object.
 * Este método se usa en HashMap al intentar determinar si una clave es
 * igual a alguna de las claves de la tabla.
 * De nuevo por defecto, el Object.equals() compara las direcciones, por
 * lo que ambas versiones de Meteorologo(3) son diferentes.
 */

/*
 * DetectorPrimaveraDos.java
 *
 * Solución al ejercicio 2 del examen de Septiembre de 2004 de
 * Fundamentos de Telemática.
 */

import java.util.*;

class Meteorologo {
    int numMeteorologo;

    Meteorologo (int n){
        numMeteorologo = n;
    }
    public int hashCode(){
        return numMeteorologo;
    }
    public boolean equals(Object o){
        return(o instanceof Meteorologo)&&(numMeteorologo ==
            ((Meteorologo)o).numMeteorologo);
    }
}

/** Devuelve una prediccion en funcion del numero aleatorio */
class Prediccion{
    boolean oscurecer = Math.random() > 0.5;

    /** @return Una prediccion */
    public String toString(){
        if(oscurecer)
            return "Seis semanas más de invierno!!!";
        else
            return "primavera temprana";
    }
}

/** Muestra la prediccion del metereologo 3
 * Asigna una prediccion a cada meteorologo y la muestra por pantalla.
 * Contiene el método main
 */
public class DetectorPrimaveraDos{

    /**
     * @param args Argumentos de la línea de comandos

```

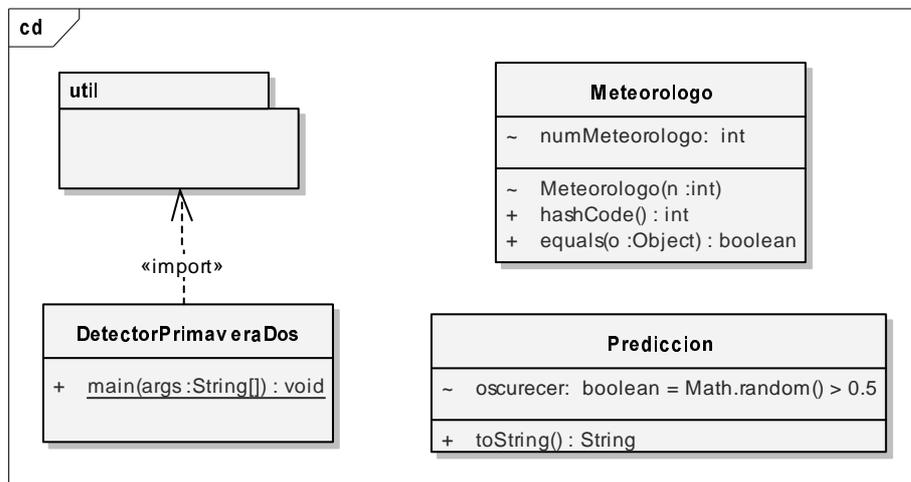
```

*/
public static void main(String args[]){
    HashMap hm = new HashMap();
    for(int i = 0; i < 10; i++){
        hm.put(new Meteorologo(i),new Prediccion());

    System.out.println("hm = " + hm +"\n");
    System.out.println("Buscando prediccion meteorologo #3");
    Meteorologo gh = new Meteorologo(3);

    if(hm.containsKey(gh))
        System.out.println((Prediccion)hm.get(gh));
    else
        System.out.println("Clave no encontrada: "+gh);
    }
}

```



Class *DetectorPrimaveraDos*

java.lang.Object

└─ DetectorPrimaveraDos

```
public class DetectorPrimaveraDos
extends java.lang.Object
```

Solución al ejercicio 2 del examen de Septiembre de 2004 de Fundamentos de Telemática.

Constructor Summary

[DetectorPrimaveraDos\(\)](#)

Method Summary

static void [main](#)(java.lang.String[] args)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DetectorPrimaveraDos

```
public DetectorPrimaveraDos()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

Parameters:

args - Argumentos de la línea de comandos

Ejercicio 1 (15/6/2005 y 8/9/2005) e/s, excepciones

La división de *tokens* de un texto de entrada es una aplicación común, y el paquete `java.io` proporciona la clase `StreamTokenizer` para este propósito. Esta clase toma un flujo de entrada (`Reader`), lo analiza y lo divide en *tokens*, permitiendo que cada uno de los *tokens* se lean secuencialmente mediante el método `nextToken`.

Cada byte del flujo de entrada es un carácter en el rango `\u0000` y `\u00FF`. El valor del carácter se usa para mirar cinco posibles atributos del carácter: *espacio en blanco*, *alfabético*, *numérico*, *comilla de string*, y *carácter de comentario*. Cada carácter puede tener cero o más de estos atributos.

Un *stream* se divide en *tokens* creando un `StreamTokenizer` con un objeto `Reader` como fuente, y estableciendo los parámetros de la exploración. El bucle de exploración utilizará `nextToken` (`int nextToken()`), que devuelve el tipo del siguiente *token* del *stream*.

Cuando el método `nextToken()` reconoce un *token*, devuelve un valor asociado a un tipo, y este valor también es puesto en el campo `ttype`. Hay cuatro tipos de *tokens* (son los valores que puede devolver este método). Los valores que puede devolver este método se mantienen en la clase como valores estáticos, y son los siguientes:

- **TT_WORD**: Cuando se ha explorado una palabra. El campo `sval` de tipo `String` contiene la palabra encontrada.
- **TT_NUMBER**: Cuando se ha explorado un número. El campo `nval` de tipo `double` contiene el valor del número. Solo se reconocen números decimales en coma flotante (con o sin coma decimal). El divisor en *tokens* no entiene `3.4e79` como número en coma flotante, ni `0xffff` como número hexadecimal.
- **TT_EOL**: Cuando se ha encontrado un final de línea.
- **TT_EOF**: Cuando se ha encontrado un final de archivo.

Se pide:

- Implementar el código Java de una clase con un método que sume de todos los valores numéricos obtenidos de los *tokens* numéricos del flujo de entrada y lo devuelva en el nombre del método.
- Implemente otra clase con el método `main` de forma que realice la llamada al método implementado y muestre el resultado por pantalla, la suma de los valores numéricos leídos de un fichero. ¿Qué clases necesita para ello? ¿Cuál será el constructor a utilizar?

Nota: El código de los dos apartados deben de ser coherente uno con el otro.

Algunos atributos de <code>StreamTokenizer</code>	Explicación
<code>double nval</code>	Si el actual <i>token</i> es un número, este campo contiene el valor.
<code>String sval</code>	Si el actual <i>token</i> es una palabra, este campo contiene el valor.
<code>int ttype</code>	Después de la llamada a <code>nextToken</code> , <code>ttype</code> contiene el tipo del <i>token</i> que se acaba de leer.

El constructor de la clase es: `StreamTokenizer(Reader is)`.

Solución del Ejercicio 1

```
/*
 * Tokens.java
 *
 * Solución al ejercicio 1 del examen de Septiembre de 2005 de
 * Fundamentos de Telemática.
 */

import java.io.FileReader;
import java.io.StreamTokenizer;
import java.io.IOException;
import java.io.FileNotFoundException;

/**
 * Clase Tokens que contiene a los métodos suma Valores y leerAtributos
 */
public class Tokens{

    /**
     * Método sumaValores que suma todos los valores numéricos obtenidos
     * de los tokens numéricos del flujo de entrada
     */
    public double sumaValores(StreamTokenizer is) throws IOException{
        double resultado = 0.0;
        is.commentChar('#');
        is.ordinaryChar('/');

        while( is.nextToken() != StreamTokenizer.TT_EOF){
            if(is.ttype == StreamTokenizer.TT_NUMBER){
                resultado += is.nval;
                System.out.println("is.nval = "+is.nval);
                System.out.println("resultado = "+resultado );
                System.out.println(" ");
            }else if(is.ttype == StreamTokenizer.TT_WORD){

                System.out.println("="+is.sval);
            }
        }
        return resultado;
    }

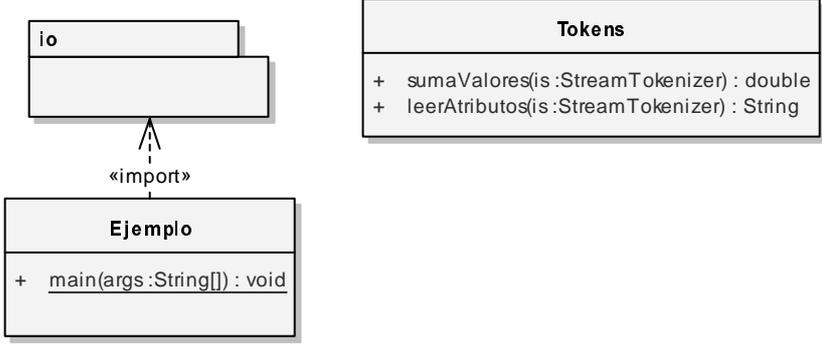
    public String leerAtributos(StreamTokenizer is) throws IOException{
        return this.toString();
    }
}

/**
 * Clase Ejemplo que contiene el método Main
 */
class Ejemplo{
    public static void main(String args[]){
        Tokens o = new Tokens();
        FileReader archivo = null;
        try{
            archivo = new FileReader("elarchivo.txt");
            StreamTokenizer in = new StreamTokenizer (archivo);

            System.out.println ("El resultado = "+o.sumaValores(in));

        }catch(FileNotFoundException e){
        }catch(IOException e){
        }
    }
}
```

cd



Class Tokens

java.lang.Object
└ **Tokens**

```
public class Tokens  
extends java.lang.Object
```

Solución al ejercicio 1 del examen de Septiembre de 2005 de Fundamentos de Telemática

Constructor Summary

Tokens ()

Method Summary

java.lang.String	leerAtributos (java.io.StreamTokenizer is)
double	sumaValores (java.io.StreamTokenizer is) Método sumaValores que suma todos los valores numéricos obtenidos de los tokens numéricos del flujo de entrada

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Tokens

```
public Tokens()
```

Method Detail

sumaValores

```
public double sumaValores(java.io.StreamTokenizer is)  
throws java.io.IOException
```

Método sumaValores que suma todos los valores numéricos obtenidos de los tokens numéricos del flujo de entrada

Throws:

java.io.IOException

leerAtributos

```
public java.lang.String leerAtributos(java.io.StreamTokenizer is)  
throws java.io.IOException
```

Throws:

java.io.IOException

Ejercicio 2 (15/6/2005 y 8/9/2005) java.util, excepciones

En el siguiente ejercicio se codifica en Java una clase que implementa un iterador que filtra cadenas de texto (`String`) de longitud mayor que una dada. Los iteradores son objetos útiles, y podemos desear escribir los nuestros, incluso aunque no estemos implementando un tipo de colección.

- Codifique en el lenguaje Java una clase que implemente los métodos del iterador para que filtre las cadenas que tenga una longitud superior a un determinado valor. Use un constructor al que se le pasa un iterador y una dimensión máxima para el filtrado de los objetos `String`.
- Diagrama de clases UML de la clase implementada en el ejercicio 2. Hagalo de forma detallada.

<i>Métodos de Iterator</i>
<code>public boolean hasNext()</code>
<code>public Object next()</code>
<code>public void remove()</code>

El método `remove()`, implementa una operación que no se va a utilizar. Deberá lanzar la excepción `UnsupportedOperationException` (esta excepción pertenece al paquete `java.lang`).

El método `next()` generará la excepción `NoSuchElementException` (del paquete `java.util`) si se invoca y no hay más elementos en el iterador.

El método `length()` da la longitud de un objeto `String`.

Solución del Ejercicio 2

```
/*
 * StringCortos.java
 *
 * Solución al ejercicio 2 del examen de Septiembre de 2005 de
 * Fundamentos de Telemática.
 */

import java.util.Iterator;
import java.util.ArrayList;
import java.util.NoSuchElementException;

/**
 * Clase StringCortos que implementa la interfaz Iterator
 */
public class StringCortos implements Iterator{

    private Iterator    cadenas;
    private String      sigCortos;
    private final int  maxLong;

    public StringCortos (Iterator cadenas,int maxLong){

        this.cadenas = cadenas;
        this.maxLong = maxLong;
        this.sigCortos = null;
    }

    public boolean hasNext(){

        if(this.sigCortos != null)
            return true;

        while (cadenas.hasNext()){
            this.sigCortos= (String)cadenas.next();
            if(this.sigCortos.length() <= maxLong)
                return true;
        }
        this.sigCortos = null;
        return false;
    }

    public Object next(){

        if((this.sigCortos == null) && (!this.hasNext()))
            throw new NoSuchElementException();

        String n = sigCortos;
        sigCortos = null;
        return n;
    }

    public void remove(){
        throw new UnsupportedOperationException();
    }
}
```

```

/**
 * Clase Ejemplo que contiene el método Main
 */
class Ejemplo{

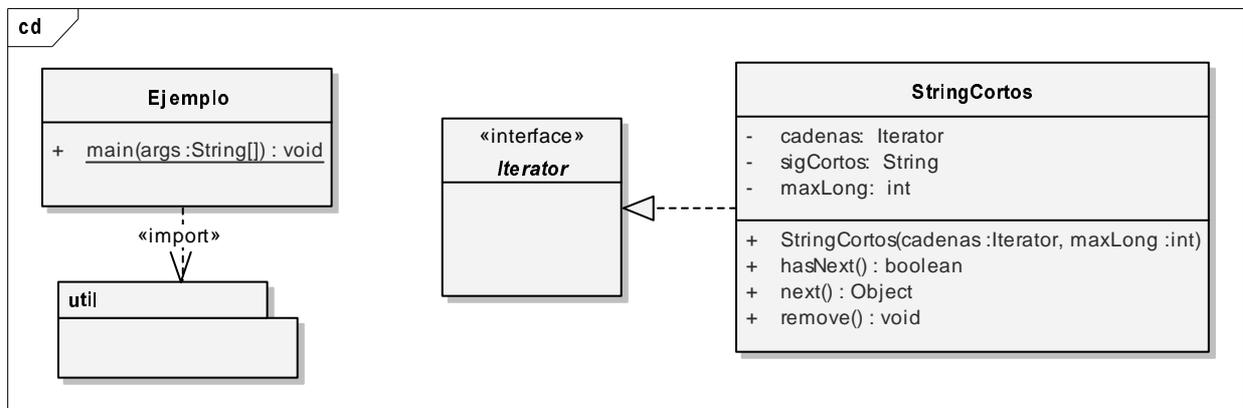
    /**
     * @param args Argumentos de la línea de comandos
     */
    public static void main(String args[]){

        ArrayList c = new ArrayList();
        c.add("UnoUNOjjj");
        c.add("DosDos");
        c.add("TresTresTres");
        c.add("CuatroCuatro");
        c.iterator();

        StringCortos s = new StringCortos (c.iterator(),10);

        while(s.hasNext()){
            System.out.println((String)s.next());
        }
    }
}

```



Class StringCortos

java.lang.Object
└─ StringCortos

All Implemented Interfaces:

java.util.Iterator

```
public class StringCortos
extends java.lang.Object
implements java.util.Iterator
```

Solución al ejercicio 2 del examen de Septiembre de 2005 de Fundamentos de Telemática

Constructor Summary

StringCortos(java.util.Iterator cadenas, int maxLong)

Method Summary

boolean	hasNext ()
java.lang.Object	next ()
void	remove ()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

StringCortos

```
public StringCortos(java.util.Iterator cadenas,
                    int maxLong)
```

Method Detail

hasNext

```
public boolean hasNext()
```

Specified by:
hasNext in interface `java.util.Iterator`

next

public java.lang.Object **next**()

Specified by:
next in interface `java.util.Iterator`

remove

public void **remove**()

Specified by:
remove in interface `java.util.Iterator`

[Package](#) **[Class](#)** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Ejercicio 1 (4/2/2006, 25/5/2006 y ?/9/2006) e/s

Un flujo `Pushback` nos permite enviar de vuelta, o “devolver” caracteres o bytes cuando se “han leído demasiados”, o cuando la aplicación lo requiera. `Pushback` es generalmente útil para dividir la entrada en *tokens*. Por ejemplo, los exploradores léxicos a menudo sólo saben que un *token* ha terminado (como por ejemplo un identificador) cuando leen el primer carácter que lo sigue. Una vez visto ese carácter, el explorador debe devolverlo al flujo de entrada, para que quede disponible como el primer carácter del siguiente *token*.

Este ejercicio usa un flujo `PushbackInputStream` para indicar la secuencia consecutiva más larga con el mismo byte de entrada. Los resultados los muestra por pantalla.

```
import java.io.IOException;
import java.io.PushbackInputStream;

class Contarsecuencia{
    public static void main(String[] args) throws IOException{
        PushbackInputStream in = new PushbackInputStream (System.in);
        int max = 0; // secuencia más larga encontrada
        int maxB= -1; // el byte de esa secuencia
        int b; // byte actual de la entrada

        do{
            int cnt;
            int bl = in.read(); // primer byte de la secuencia
            for(cnt = 1; (b = in.read()) ==_____ ; cnt++)
                ;
            if(cnt > max){
                max = cnt; //recuerda la longitud
                maxB= bl; //recuerda el valor del byte
            }
            _____ //devuelve el inicio de la siguiente
sec.
        }while(b != -1); //hasta el final de la entrada.
        System.out.println(max + " bytes de " + (char)maxB);
    }
}
```

- Ponga el código que falta en las zonas marcadas para que el programa devuelva la secuencia de caracteres más larga y el carácter de la secuencia más larga.

Todos los flujos `Pushback` tienen tres variantes de `unread`, estos aparecen a continuación:

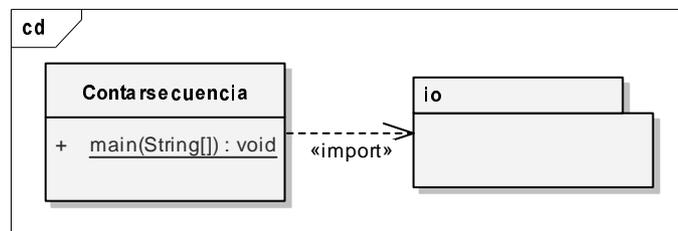
<code>public void unread(int c) throws IOException</code> Devuelve el carácter <code>c</code> . Si el buffer de devolución está lleno se lanza una excepción <code>IOException</code> .
<code>public void unread(char[]buf,int desp, int cuenta) throws IOException</code> Devuelve los caracteres del subarray especificado.
<code>public void unread(char[]buf) throws IOException</code> Equivalente a <code>unread(buf,0,buf.length)</code> .

Solución al Ejercicio 1

```
import java.io.IOException;
import java.io.PushbackInputStream;

class Contarsecuencia{
    public static void main(String[] args) throws IOException{
        PushbackInputStream in = new PushbackInputStream (System.in);
        int max = 0; // secuencia más larga encontrada
        int maxB= -1; // el byte de esa secuencia
        int b; // byte actual de la entrada

        do{
            int cnt;
            int b1 = in.read(); // primer byte de la secuencia
            for(cnt = 1; (b = in.read()) == b1 ; cnt++)
                ;
            if(cnt > max){
                max = cnt; //recuerda la longitud
                maxB= b1; //recuerda el valor del byte
            }
            in.unread(b); //devuelve el inicio de la siguiente sec.
        }while(b != -1); //hasta el final de la entrada.
        System.out.println(max + " bytes de " + (char)maxB);
    }
}
```



Class Contarsecuencia

java.lang.Object

Contarsecuencia

class **Contarsecuencia**
extends java.lang.Object

Constructor Summary

(package private)	<u>Contarsecuencia()</u>
-------------------	--

Method Summary

static void	<u>main</u> (java.lang.String[] args)
-------------	---

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Contarsecuencia

Contarsecuencia()

Method Detail

main

```
public static void main(java.lang.String[] args)
                    throws java.io.IOException
```

Throws:

java.io.IOException

Ejercicio 2 (4/2/2006, 25/5/2006 y ?/9/2006)

Se desea comprobar si los caracteres que tiene un `String`, siguen el patrón determinado (expresado mediante unas *reglas*) por los caracteres de otro `String`. Para realizarlo se va a implementar la clase `Patrones`, que contenga el método `comprueba`, que realice esta operación.

El método (`public boolean comprueba (String patron, String secuencia)`), necesita dos objetos de tipo `String`, uno el patrón al que debe ajustarse el segundo de los parámetros. Se debe invocar de la siguiente forma,

```
Patrones p = new Patrones ();
boolean resultado = p.comprueba ("A*BC", "AAAAABC");
```

El valor devuelto es un valor booleano, `true` si la secuencia (*secuencia*) se ajusta al patrón (*patron*), o `false` si la secuencia no se ajusta al patrón. Se supondrá que los patrones (*patron*) que se pasan como parámetro en el primer argumento son siempre correctos.

Las *reglas* para establecer los patrones es de la siguiente forma.

Patrón	Explicación
A?	A puede aparecer una o ninguna vez. Ejemplo: Devuelve <code>true</code> , <code>p.comprueba ("A?BC", "BC")</code> ;
A+	A puede aparecer una o más veces. Ejemplo: Devuelve <code>true</code> , <code>p.comprueba ("A+BC", "AABC")</code> ;
A*	A puede aparecer cero o más veces. Ejemplo: Devuelve <code>true</code> , <code>p.comprueba ("A*BC", "BC")</code> ;
A{n}	A debe aparecer exactamente n veces (siendo n un valor entero). Ejemplo: Devuelve <code>true</code> , <code>p.comprueba ("A{3}Z", "AAAZ")</code> ;
A{n,}	A debe aparecer al menos n veces (con n entero). Ejemplo: Devuelve <code>true</code> , <code>p.comprueba ("A{3,}Z", "AAAZ")</code> ;
A{n,m}	A debe aparecer al menos n veces pero no más de m (n, m enteros). Ejemplo: Devuelve <code>true</code> , <code>p.comprueba ("A{3,5}Z", "AAAAZ")</code> ;

- Implemente clase `Patrones`.

Para realizarlo puede utilizar la siguiente interfaz:

```
public interface Ctes{
    public static final char abreLlave      = '{';
    public static final char cierraLlave   = '}';
    public static final char asterisco     = '*';
    public static final char mas           = '+';
    public static final char interrogacion  = '?';
    public static final char coma          = ',';
}
```

Sugerencia, puede usar las siguientes variables de instancia, para la clase `Patrones`:

```
private int cursorPatron      = 0;
private int cursorSecuencia  = 0;
private String patron         = null;
private String secuencia     = null;
```

Métodos de String	<code>public int length()</code> , Devuelve la longitud de un string.
	<code>String substring(int beginIndex)</code> <code>String substring(int beginIndex, int endIndex)</code> Devuelve un nuevo String que es un substring del String .
	<code>public char charAt(int indice)</code> Devuelve el carácter que hay en el String , en la posición que se le pasa como parámetro.
	<code>public int indexOf(int ch)</code> , <code>public int indexOf(int ch, int fromIndex)</code> , <code>public int indexOf(String str)</code> , <code>public int indexOf(String str, int fromIndex)</code> . Devuelve el índice del String con la primera aparición de la ocurrencia especificada como parámetro.
Método de Character	<code>public static boolean isLetter(char ch)</code> Determina si el carácter especificado es una letra.
Método de Integer	<code>public static int parseInt(String s)</code> Devuelve como entero el número en forma de cadena.


```

        m = Integer.parseInt((
            patron.substring(coma+1,llave));
    }
    cursorPatron = llave+1;
}
Error = verifica (n, m, ElPatron);
ElPatron = 0;
break;
default:
    cursorPatron++;
    if(Character.isLetter(charPatron)){
        ElPatron = charPatron;
        if((cursorPatron) == patron.length()
            || (Character.isLetter(patron.charAt(cursorPatron)) )){
            Error = verifica (1,1, ElPatron );
            ElPatron = 0;
        }
    }
}
}
return Error ;
}
private boolean verifica(int n, int m, char ElPatron){
    boolean tmp = false;
    int contador = 0;

    for (contador = 0;(cursorSecuencia < secuencia.length())
        && (ElPatron == secuencia.charAt(cursorSecuencia))
        ; contador++ , cursorSecuencia++)
        ;

    if((n <= contador)&&((-1 == m)|| (contador <= m)))
        tmp = true;
    return tmp ;
}
public void reset(){
    this.cursorPatron      = this.cursorSecuencia = 0;
    this.patron            = this.secuencia      = null;
}
}

class prueba{
    public static void main(String args[]){
        Patrones p = new Patrones();
        p.reset();
        System.out.println("Resultado = "+p.comprueba("A+BC", "AABC"));
        p.reset();
        System.out.println("Resultado = "+p.comprueba("A?BC", "BC"));
        p.reset();
        System.out.println("Resultado = "+p.comprueba("A*BC", "BC"));
        p.reset();
        System.out.println("Resultado = "+p.comprueba("A{3}Z", "AAAZ"));
        p.reset();
        System.out.println("Resultado = "+p.comprueba("A{3,}Z", "AAAZ")); //
        p.reset();
        System.out.println("Resultado = "+p.comprueba("A{3,5}Z", "AAAAZ"));
    }
}
}

```

cd

```
«interface»  
Ctes  
+ abreLlave: char = '{'  
+ cierraLlave: char = '}'  
+ asterisco: char = '*'  
+ mas: char = '+'  
+ interrogacion: char = '?'  
+ coma: char = ','
```

```
prueba  
+ main(String[]) : void
```

```
Patrones  
- cursorPatron: int = 0  
- cursorSecuencia: int = 0  
- patron: String = null  
- secuencia: String = null  
+ comprueba(String, String) : boolean  
- verifica(int, int, char) : boolean  
+ reset() : void
```

Class Patrones

java.lang.Object

Patrones

```
public class Patrones
extends java.lang.Object
```

Field Summary

private int	<u>cursorPatron</u>
private int	<u>cursorSecuencia</u>
private java.lang.String	<u>patron</u>
private java.lang.String	<u>secuencia</u>

Constructor Summary

Patrones ()

Method Summary

boolean	<u>comprueba</u> (java.lang.String patronX, java.lang.String secuenciaX)
void	<u>reset</u> ()
private boolean	<u>verifica</u> (int n, int m, char ElPatron)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

cursorPatron

private int cursorPatron

cursorSecuencia

private int cursorSecuencia

patron

private java.lang.String patron

secuencia

```
private java.lang.String secuencia
```

Constructor Detail

Patrones

```
public Patrones()
```

Method Detail

comprueba

```
public boolean comprueba(java.lang.String patronX,  
                           java.lang.String secuenciaX)
```

verifica

```
private boolean verifica(int n,  
                           int m,  
                           char ElPatron)
```

reset

```
public void reset()
```

Ejercicio 3 (7/9/2006)

Implemente en el lenguaje de programación Java una clase pública (**SubPatron**) en el paquete **utilidades** contenga las siguientes variables de instancia privadas:

- o **subpatron** de tipo **String**
- o **minOccurs** de tipo **int**
- o **maxOccurs** de tipo **int**
- o **value** de tipo **int**

Las tres primeras variables (**subpatron**, **minOccurs** y **maxOccurs**) se configuran en el constructor, y la última se configura con un método el método **setValue**.

Implemente los métodos **get**, para todas las variables de instancia.
Dibuje el diagrama de la clases de la clase implementada.

Métodos de String
public int length() , Devuelve la longitud de un string.
String substring(int beginIndex) String substring(int beginIndex, int endIndex) Devuelve un nuevo String que es un substring del String .
public char charAt(int indice) Devuelve el carácter que hay en el String , en la posición que se le pasa como parámetro.
public int indexOf(int ch) , public int indexOf(int ch, int fromIndex) , public int indexOf(String str) , public int indexOf(String str, int fromIndex) . Devuelve el índice del String con la primera aparición de la ocurrencia especificada como parámetro.

Método de Character	public static boolean isLetter(char ch) Determina si el carácter especificado es una letra.
----------------------------	--

Método de Integer	public static int parseInt(String s) Devuelve como entero el número en forma de cadena.
--------------------------	--

Solución al Ejercicio 3

```
package utilidades;

public class SubPatron{

    private String subpatron;
    private int minOccurs;
    private int maxOccurs;
    private int value;

    public SubPatron(String subpatron, int minOccurs,
                     int maxOccurs){
        this.subpatron = subpatron;
        this.minOccurs = minOccurs;
        this.maxOccurs = maxOccurs;
    }

    public void setValue(int n){
        value = n;
    }

    public String getSubpatron(){
        return this.subpatron;
    }

    public int getMinOccurs(){
        return this.minOccurs;
    }

    public int getMaxOccurs(){
        return this.maxOccurs;
    }

    private int getValue(){
        return this.value;
    }

}
```

cd

utilidades

SubPatron

- subpatron: String
- minOccurs: int
- maxOccurs: int
- value: int

+ SubPatron(subpatron :String, minOccurs:int, maxOccurs:int)
+ setValue(n :int) : void
+ getSubpatron() : String
+ getMinOccurs() : int
+ getMaxOccurs() : int
- getValue() : int

utilidades

Class SubPatron

java.lang.Object

utilidades.SubPatron

```
public class SubPatron
extends java.lang.Object
```

La clase Subpatron

Field Summary

private int	<u>maxOccurs</u>
private int	<u>minOccurs</u>
private java.lang.String	<u>subpatron</u>
private int	<u>value</u>

Constructor Summary

<u>SubPatron</u> (java.lang.String subpatron, int minOccurs, int maxOccurs) Constructor con tres parametros
--

Method Summary

int	<u>getMaxOccurs</u> () Devuelve la variable de instancia maxOccurs
int	<u>getMinOccurs</u> () Devuelve la variable de instancia minOccurs
java.lang.String	<u>getSubpatron</u> () Devuelve la variable de instancia subpatron
private int	<u>getValue</u> () Devuelve la variable de instancia value
void	<u>setValue</u> (int n) Configura la variable de instancia value

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

subpatron

```
private java.lang.String subpatron
```

minOccurs

```
private int minOccurs
```

maxOccurs

```
private int maxOccurs
```

value

```
private int value
```

Constructor Detail

SubPatron

```
public SubPatron(java.lang.String subpatron,  
                 int minOccurs,  
                 int maxOccurs)
```

Constructor con tres parametros

Method Detail

setValue

```
public void setValue(int n)
```

Configura la variable de instancia value

getSubpatron

```
public java.lang.String getSubpatron()
```

Devuelve la variable de instancia subpatron

getMinOccurs

```
public int getMinOccurs()
```

Devuelve la variable de instancia minOccurs

getMaxOccurs

```
public int getMaxOccurs()
```

Devuelve la variable de instancia maxOccurs

getValue

```
private int getValue()
```

Devuelve la variable de instancia value
