# Acceso a ficheros

# <https://datos.codeandcoke.com/apuntes:ficheros>

## Tipos de ficheros

Desde el punto de vista de un programador solamente distinguiremos entre dos tipos de ficheros:

* **Ficheros de texto** cuando el contenido del fichero contenga exclusivamente caracteres de texto (podemos leerlo con un simple editor de texto)

| **Extensión** | **Tipo de fichero** |
| --- | --- |
| .txt | Fichero de texto plano |
| .xml | Fichero XML |
| .json | Fichero de intercambio de información |
| .props | Fichero de propiedades |
| .conf | Fichero de configuración |
| .sql | Script SQL |
| .srt | Fichero de subtítulo |

* **Ficheros binarios** cuando no estén compuestos exclusivamente de texto. Pueden contener imágenes, videos, ficheros, . . . aunque también podemos considerar un fichero binario a un fichero de Microsoft Word en el que sólo hayamos escrito algún texto puesto que, al almacenarse el fichero, el procesador de texto incluye alguna información binaria

| **Extensión** | **Tipo de fichero** |
| --- | --- |
| .pdf | Fichero PDF |
| .jpg | Fichero de imagen |
| .doc, .docx | Fichero de Microsoft Word |
| .avi | Fichero de video |
| .ppt, .pptx | Fichero de PowerPoint |

A veces, en ficheros binarios, podremos encontrarnos con las extensiones *.bin* o *.dat* para hacer referencia a ficheros que contienen información binaria en un formato que no está ampliamente difundido. Serán simplemente ficheros que una aplicación determinada es capaz de leer/escribir de una forma específica solo definida para dicha aplicación.

## Propiedades del Sistema

En cualquier caso, para el acceso a ficheros independientemente del tipo de los mismos, conviene conocer el funcionamiento de las [System Properties](https://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html) de las que dispone Java en su API. Éstas permiten acceder a propiedades de la configuración del sistema y, entre otras, podemos encontrarnos con algunas muy interesantes relacionadas con este tema:

* “file.separator” Obtiene el caracter, según el S.Operativo, para la separación de las rutas (/ ó \). También se puede utilizar la constante File.separator
* “user.home” Obtiene la ruta de la carpeta personal del usuario (que dependerá del S.Operativo en casa caso)
* “user.dir” Obtiene la ruta en la que se encuentra actualmente el usuario
* “line.separator” Obtiene el caracter que separa las líneas de un fichero de texto (difiere entre Windows/Linux)

En el caso de que se quiera acceder al valor de alguna de estas propiedades debe hacerse utilizando la llamada al método System.getProperty(String)

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("La carpeta de mi usuario es " + [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).getProperty("user.home"));

## Ficheros de texto

En esta parte vamos a trabajar con 3 tipos de ficheros de texto:

* **Ficheros de texto plano** que contendrán texto *libre* y donde podremos escribir sin respetar ningún tipo de formato
* **Ficheros de configuración** que contendrán información de configuración para una aplicación. Tienen un formato específico y Java además proporciona una API para trabajar más cómodamente con ellos
* **Ficheros XML** que contienen información y acompañada de etiquetas que le dan significado. Tienen unas reglas y formato más o menos definido y Java proporciona una API para trabajar con ellos
* **Ficheros JSON** que contienen información y acompañada de caracteres etiquetas que le dan significado. Tienen unas reglas y formato más o menos definido y Java proporciona una API para trabajar con ellos
* **Ficheros CSV** que contienen información separados con comas, punto y comas, … (u otros caracteres a convenir) que le dan significado de campos y registros.

## Ficheros de texto plano

Los ficheros de texto son aquellos que únicamente contienen texto, por lo que pueden ser editados directamente con cualquier editor de texto plano (Bloc de Notas, notepad++, . . .). Se podría decir que son aquellos ficheros que podrían ser *leídos* por cualquier persona. Son aquellos que normalmente se almacenan con la extensión *.txt* pero también podríamos incluir los scripts SQL (*.sql*), ficheros de código Java (*.java*), ficheros de configuración (*.ini*, *.props*, *.conf*, . . .), . . .

También se incluyen en la categoría de ficheros de texto los que además incluyen información adicional (siempre en forma de texto) que permiten interpretar los datos del fichero de una manera u otra, añadiendo más información al mismo. Estos formatos son HTML, XML, JSON, . . .

En cualquier caso, desde Java siempre se podrán leer/escribir de la misma manera, según veremos a continuación. También veremos como pueden leerse/escribirse utilizando librerías aquellos ficheros de texto plano que contienen formato, como hemos comentado anteriormente, haciendo de esa forma mucho más fácil el trabajo.

**Escribir ficheros de texto plano**

[FileWriter](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+filewriter) fichero = **null**;

[PrintWriter](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+printwriter) escritor = **null**;

try {

fichero = new [FileWriter](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+filewriter)("archivo.txt");

escritor = new [PrintWriter](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+printwriter)(fichero) ;

escritor.println("Esto es una linea del fichero");

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) {

ioe.printStackTrace() ;

} finally {

if (fichero != **null**)

try {

fichero.close();

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) { . . . }

}

**Leer ficheros de texto plano**

[File](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+file) fichero = **null**;

[FileReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+filereader) lector = **null**;

[BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader) buffer = **null**;

try {

buffer = new [BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader)(new [FileReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+filereader)(new [File](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+file)("archivo.txt")));

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) linea = **null**;

while ((linea = buffer.readLine()) != **null**)

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(linea);

} catch ([FileNotFoundException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+filenotfoundexception) fnfe) {

fnfe.printStackTrace();

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) {

ioe.printStackTrace();

} finally {

if (buffer != **null**)

try {

buffer.close();

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) { . . . }

}

## Ficheros de configuración

En la API de Java se incluyen librerías para trabajar con los ficheros de configuración. Puesto que todos siguen un mismo patrón, es la librería la que se encarga de acceder al fichero a bajo nivel y el programador sólo tiene que indicar a que propiedad quiere acceder o que propiedad quiere modificar, sin tener que añadir nada de código para leer o escribir el fichero tal y como hemos visto en el punto anterior.

Un ejemplo de fichero de configuración de esta librería de java sería el fichero que sigue:

[configuracion.props](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=3)

*# Fichero de configuracion*

*# Thu Nov 14 10:49:39 CET 2013*

user=usuario

password=micontrasena

server=localhost

port=3306

**Escribir ficheros de configuración**

Así, si queremos generar, desde Java, un fichero de configuración como el anterior:

. . .

[Properties](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+properties) configuracion = new [Properties](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+properties)();

configuracion.setProperty("user", miUsuario);

configuracion.setProperty("password", miContrasena);

configuracion.setProperty("server", elServidor);

configuracion.setProperty("port", elPuerto);

try {

configuracion.store(new [FileOutputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+fileoutputstream)("configuracion.props"),

"Fichero de configuracion");

} catch ([FileNotFoundException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+filenotfoundexception) fnfe ) {

fnfe.printStackTrace();

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) {

ioe.printStackTrace();

}

. . .

**Leer ficheros de configuración**

A la hora de leerlo, en vez de tener que recorrer todo el fichero como suele ocurrir con los ficheros de texto, simplemente tendremos que cargarlo e indicar de qué propiedad queremos obtener su valor (getProperty(String)).

. . .

[Properties](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+properties) configuracion = new [Properties](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+properties)();

try {

configuracion.load(new [FileInputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+fileinputstream)("configuracion.props"));

usuario = configuracion.getProperty("user");

password = configuracion.getProperty("password");

servidor = configuracion.getProperty("server");

puerto = [Integer](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+integer).valueOf(configuration.getProperty("port"));

} catch ([FileNotFoundException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+filenotfoundexception) fnfe ) {

fnfe.printStackTrace();

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) {

ioe.printStackTrace();

}

. . .

Para ambos casos, escribir y leer este tipo de ficheros, hay que tener en cuenta que, al tratarse de ficheros de texto, toda la información se almacena como si de un String se tratara. Por tanto, todos aquellos tipos Date, boolean o incluso cualquier tipo numérico serán almacenados en formato texto. Así, habrá que tener en cuenta las siguientes consideraciones:

* Para el caso de las fechas, deberán ser convertidas a texto cuando se quieran escribir y nuevamente reconvertidas a Date cuando se lea el fichero y queramos trabajar con ellas
* Para el caso de los tipos boolean, podemos usar el método String.valueOf(boolean) para pasarlos a String cuando queramos escribirlos. En caso de que queramos leer el fichero y pasar el valor a tipo boolean podremos usar el método Boolean.parseBoolean(String)
* Para el caso de los tipos numéricos (integer, float, double) es muy sencillo ya que Java los convertirá a String cuando sea necesario al escribir el fichero. En el caso de que queramos leerlo y convertirlos a su tipo concreto, podremos usar los métodos Integer.parseInt(String),  Float.parseFloat(String)  y Double.parseDouble(), según proceda

## Ficheros XML

Los ficheros XML permiten el intercambio de información entre aplicaciones utilizando para ello un fichero de texto plano al que se le pueden añadir etiquetas para darle significado a cada uno de los valores que se almacenan.

Por ejemplo, el siguiente fichero XML podría ser el resultado de volcar una Base de Datos sobre productos de una compañia, de forma que dicha información podría ahora leerse desde otra aplicación e incorporarla. Se puede ver como a parte de los datos de dichos productos (sus nombres, precios, . . .) aparece otra información en forma de etiquetas (entre los caracteres < y >) que permite dar significado a cada dato almacenado en el fichero. Así, podemos saber de qué estamos hablando y a qué corresponde cada valor.

Obviamente, tal y como ocurría con los ficheros de configuración, toda la información se tiene que pasar a texto para poder crear el fichero y reconvertida a su tipo original cuando se cargue de nuevo. Las mismas consideraciones que hemos tenido en cuenta antes nos servirán ahora (ver [aqui](http://datos.codeandcoke.com/apuntes:ficheros#ficheros_de_configuracion))

[productos.xml](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=6)

<?xml version="1.0" encoding="UTF-8" standalone="no">

<xml>

<productos>

<producto>

<nombre>Cereales</nombre>

<precio>3.45</precio>

</producto>

<producto>

<nombre>Colacao</nombre>

<precio>1.45</precio>

</producto>

<producto>

<nombre>Agua mineral</nombre>

<precio>1.00</precio>

</producto>

</productos>

</xml>

La clase Java que definiría cada uno de los objetos que se representan por el fichero XML anterior, sería la siguiente:

[Producto.java](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=7)

public class Producto {

private [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nombre;

private float precio;

public Producto([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nombre, float precio) {

this.nombre = nombre;

this.precio = precio;

}

public [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) getNombre () { return nombre; }

public void setNombre ([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nombre) { this.nombre = nombre; }

public float getPrecio () { return precio; }

public void setPrecio (float precio) { this.precio = precio; }

}

**Escribir ficheros XML**

Si ahora queremos generar el fichero XML con toda la información de una colección de objetos *Producto*, podemos utilizar el siguiente código:

. . .

documento = dom.createDocument(**null**, "xml", **null**);

[Element](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+element) raiz = document.createElement("productos");

documento.getDocumentElement().appendChild(raiz);

[Element](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+element) nodoProducto = **null** , nodoDatos = **null** ;

Text texto = **null**;

for (Producto producto : listaProductos) {

nodoProducto = documento.createElement("producto");

raiz.appendChild(nodoProducto);

nodoDatos = documento.createElement("nombre");

nodoProducto.appendChild(nodoDatos);

texto = documento.createTextNode(producto.getNombre());

nodoDatos.appendChild(texto);

nodoDatos = documento.createElement("precio");

nodoProducto.appendChild(nodoDatos);

texto = documento.createTextNode(producto.getPrecio());

nodoDatos.appendChild(texto);

}

. . .

**Leer ficheros XML**

Y si lo que queremos es leer un fichero XML y cargarlo como una colección Java de objetos *Producto*, el ejemplo siguiente muestra un breve ejemplo acerca de cómo acceder a la información de dicho fichero XML.

. . .

NodeList productos = documento.getElementyByTagName("producto");

for (int i = 0; i < productos.getLength(); i++) {

Node producto = productos . item ( i ) ;

[Element](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+element) elemento = ( [Element](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+element) ) producto ;

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(elemento.getElementsByTagName("nombre").item(0)

.getChildNodes().item(0).getNodeValue());

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(elemento.getElementsByTagName("precio").item(0)

.gethildNodes().item(0).getNodeValue());

}

. . .

## Ficheros binarios

[Producto.java](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=10)

public class Producto implements [Serializable](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+serializable) {

private static final long serialVersionUID = 1L;

private [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nombre;

private float precio;

public Producto([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nombre, float precio) {

this.nombre = nombre;

this.precio = precio;

}

public [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) getNombre() { return nombre; }

public void setNombre([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nombre) { this.nombre = nombre; }

public float getPrecio() { return precio ; }

public void setPrecio(float precio) { this.precio = precio; }

}

## Serialización de objetos

*Serializar* es el proceso por el cual un objeto en memoria pasa a transformarse en una estructura que pueda ser almacenada en un fichero (persistencia). Al proceso contrario le llamaremos *deserializar*.

Hay que tener en cuenta que durante el proceso de serialización, cada objeto se serializa a un fichero, por lo que si queremos almacenar todos los objetos de una aplicación, la idea más conveniente es tener todos éstos en alguna estructura compleja (y por comodidad dinámica) de forma que sea esta estructura la que serialicemos o deserialicemos para guardar o cargar los objetos de una aplicación. Estructuras como ArrayList, Set ó HashMap son clases muy utilizadas para trabajar de esta manera.

**Serializar un objeto**

. . .

[ObjectOutputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+objectoutputstream) serializador = **null**;

try {

serializador = new [ObjectOutputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+objectoutputstream)(new [FileOutputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+fileoutputstream)("archivo.dat"));

serializador.writeObject(listaProductos);

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) {

. . .

} finally {

if (serializador != **null**)

try {

serializador.close();

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) {

ioe.printStackTrace();

}

}

. . .

**Deserializar un objeto**

. . .

List<Producto> listaProductos = **null**;

[ObjectInputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+objectinputstream) deserializador = **null**;

try {

deserializador = new [ObjectInputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+objectinputstream)(new [FileInputStream](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+fileinputstream)("archivo.dat"));

listaProductos = (ArrayList<Producto>)deserializador.readObject();

} catch ([FileNotFoundException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+filenotfoundexception) fnfe ) {

fnfe.printStackTrace();

} catch ([ClassNotFoundException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+classnotfoundexception) cnfe ) {

cnfe.printStackTrace();

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) {

ioe.printStackTrace();

} finally {

if (deserializador != **null**)

try {

deserializador.close();

} catch ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) ioe) {

ioe.printStackTrace();

}

}

. . .

# Estructuras de datos

## ArrayList

[**ArrayList**](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html)**es un *array* redimensionable que implementa el *interface***[**List**](https://docs.oracle.com/javase/8/docs/api/java/util/List.html)**.** Permite todos los elementos, incluído el valor null. Como es redimensionable, dispone de método para modificar el tamaño del mismo.

Para el acceso concurrente con varios hilos, hay que tener en cuenta que esta estructura de datos no está sincronizada y su acceso deberá estarlo de forma externa. También se podría emplear la clase [Vector](https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html) que es igual que ésta pero está ya sincronizada.

**Operaciones más comunes**

* Añadir un elemento

List<String> listaCadenas = new ArrayList<>();

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nuevaCadena = "Esto es una cadena";

**listaCadenas.add(nuevaCadena);**

* Añadir un elemento en una posición determinada

int posicion = 3;

**listaCadenas.add(3, nuevaCadena);**

* Reemplazar el elemento que ocupa una posición determinada

int posicion = 3;

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) otraCadena = "Esto es otra cadena";

**listaCadenas.set(3, otraCadena);**

* Eliminar un elemento

int posicion = 3;

**listaCadenas.remove(posicion);**

* Obtener la referencia a un elemento

int posicion = 3;

[**String**](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) **unaCadena = listaCadenas.get(posicion);**

* Conocer el número de elementos

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("El ArrayList tiene " + **listaCadenas.size()** + " elementos");

* Eliminar todos los elementos de la lista

**listaCadenas.clear();**

* Añadir todos los elementos de otra lista

List<String> otraListaCadenas = new ArrayList<>();

. . .

. . .

*// Se puede pasar como parámetro un objeto que implemente el interfaz Collection*

**listaCadenas.addAll(otraListaCadenas);**

* Comprobar si esta vacía

if **(listaCadenas.isEmpty())** {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("La lista no tiene elementos");

}

* Obtener un array (estático) con todos los elemntos de la lista

[**String**](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)**[] arrayCadenas = listaCadenas.toArray();**

## HashMap

[**HashMap**](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html)**es una tabla *hash* que implementa la interface**[**Map**](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html)**.** Permite el valor null tanto para valores como para claves. Al contrario que [TreeSet](https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html) no garantiza el orden de los elementos o que éste se mantenga.

Hay que tener en cuenta que el acceso a esta implementación no está sincronizado, pero existe una implementación similar y sincronizada, que es la clase [Hashtable](https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html" \o "https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html).

**Operaciones más comunes**

* **Añadir un elemento (pareja clave-valor)**

Map<[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string), Libro> libros = new HashMap<,>();

. . .

Libro libro = new Libro();

libro.setTitulo("Secuestrado");

libro.setAutor("Robert Louis Stevenson");

. . .

libros.add(libro.getTitulo(), libro);

* **Obtener un elemento por clave**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) tituloLibro = "Secuestrado";

**Libro esteLibro = libros.get(titulo);**

* **Comprobar si existe una clave**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) titulo = "Secuestrado";

if (**libros.containsKey(titulo))** {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("El libro con el título " + titulo + " existe en tu colección");

}

* **Comprobar si esta vacío**

if **(libros.isEmpty())** {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Tu colección de libros está vacía");

}

* **Eliminar todos los elementos**

libros.clear();

* **Eliminar un elemento (por clave)**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) titulo = "Secuestrado";

**libros.remove(titulo);**

* Obtener una Collection con todos los valores

Collection<Libro> coleccionLibros = **libros.values();**

*// ArrayList, TreeSet y LinkedList implementan el interface Collection*

* Obtener un Set con todas las claves

Set<String> titulos = **libros.keySet();**

*// TreeSet y HashSet implementan el interface Set*

* Comprobar el tamaño del Map

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Tienes " + **libros.size()** + " libros en tu colección");

* Concatenar todos los elementos de otro Map

Map<[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string), Libro> masLibros = new HashMap<,>();

. . .

. . .

**libros.putAll(masLibros);**

## TreeSet

[TreeSet](https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html) es una colección que mantiene los elementos ordenados de forma natural o bien a través de un [Comparator](https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html).

**Operaciones más comunes**

* **Añadir un elemento**

Set<String> cadenas = new TreeSet<>();

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) unaCadena = "Esto es una cadena";

cadenas.add(unaCadena);

* **Añadir todos los elementos de otra Collection**

List<String> masCadenas = new ArrayList<>();

. . .

cadenas.addAll(masCadenas);

* Obtener un elemento
* Eliminar todos los elementos

**cadenas.clear();**

* Comprueba si existe un elemento en la lista

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) otraCadena = "Esto es una cadena";

if (**cadenas.contains(otraCadena))** {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("La cadena existe");

}

* **Obtener el primer elemento**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) primeraCadena = **cadenas.first();**

* **Obtener y eliminar el primer elemento**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) primeraCadena = **cadenas.pollFirst();**

* **Obtener el último elemento**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) ultimaCadena = **cadenas.last();**

* **Obtener y eliminar el último elemento**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) ultimaCadena = **cadenas.pollLast();**

* **Comprobar si esta vacío**

if (**cadenas.isEmpty())** {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("El Set de cadenas esta vacío");

}

* **Comprobar el número de elementos**

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Tienes " + **cadenas.size()** + " cadenas");

* **Obtiene la parte del Set cuyos elementos son menores que uno pasado por parámetro**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) cadenaLimite = "Una cadena";

SortedSet<String> cadenasMenores = cadenas.headSet(cadenaLimite);

*// SortedSet es el interface que implementa TreeSet*

* **Obtiene la parte del Set cuyos elementos están entre dos determinados**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) cadenaMenor = "Una cadena";

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) cadenaMayor = "Más cadena";

SortedSet<String> cadenasMenores = cadenas.subSet(cadenaMenor, cadenaMayor);

*// SortedSet es el interface que implementa TreeSet*

## TreeMap

[TreeMap](https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html) es una colección de pares *clave-valor* como [HashMap](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html) pero mantiene los elementos ordenados por su clave de forma natural o bien a través de un [Comparator](https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html).

**Operaciones más comunes**

* **Añadir un elemento (pareja clave-valor)**

Map<[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string), Libro> libros = new TreeMap<,>();

. . .

Libro libro = new Libro();

libro.setTitulo("Secuestrado");

libro.setAutor("Robert Louis Stevenson");

. . .

libros.put(libro.getTitulo(), libro);

* **Obtener un elemento**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) tituloLibro = "Secuestrado";

Libro esteLibro = libros.get(titulo);

* **Comprobar si existe una clave**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) titulo = "Secuestrado";

if (libros.containsKey(titulo)) {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("El libro con el título " + titulo + " existe en tu colección");

}

* **Comprobar si esta vacío**

if (libros.isEmpty()) {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Tu colección de libros está vacía");

}

* **Eliminar todos los elementos**

libros.clear();

* **Eliminar un elemento (por clave)**

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) titulo = "Secuestrado";

libros.remove(titulo);

* Obtener una Collection con todos los valores

Collection<Libro> coleccionLibros = **libros.values();**

*// ArrayList, TreeSet y LinkedList implementan el interface Collection*

* Obtener un Set con todas las claves

Set<String> titulos = **libros.keySet();**

*// TreeSet y HashSet implementan el interface Set*

* Comprobar el tamaño del Map

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Tienes " + **libros.size()** + " libros en tu colección");

* Concatenar todos los elementos de otro Map

Map<[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string), Libro> masLibros = new TreeMap<,>();

. . .

. . .

**libros.putAll(masLibros);**

## LinkedList

[LinkedList](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html) Es una lista doblemente enlazada que implementa el interfaz [List](https://docs.oracle.com/javase/8/docs/api/java/util/List.html)

**Operaciones más comunes**

* **Añadir un elemento**

List<Libro> listaLibros = new LinkedList<>();

. . .

Libro libro = new Libro();

libro.setTitulo("Secuestrado");

libro.setAutor("Robert Louis Stevenson");

. . .

listaLibros.add(libro);

* **Añadir un elemento al principio**

List<Libro> listaLibros = new LinkedList<>();

. . .

Libro libro = new Libro();

libro.setTitulo("Secuestrado");

libro.setAutor("Robert Louis Stevenson");

. . .

listaLibros.**addFirst(libro);**

* **Añadir un elemento al final**

List<Libro> listaLibros = new LinkedList<>();

. . .

Libro libro = new Libro();

libro.setTitulo("Secuestrado");

libro.setAutor("Robert Louis Stevenson");

. . .

listaLibros.**addLast(libro);**

* **Añadir toda una colección al final**

List<Libro> listaLibros = new LinkedList<>();

List<Libro> otraListaDeLibros = new ArrayList<>();

. . .

*// Podemos añadir cualquier objeto que implemente el interface Collection*

**listaLibros.addAll(otraListaDeLibros);**

* **Obtener un elemento por su posición**

Libro unLibro = listaLibros.get(4);

* **Obtener el primer elemento**

Libro unLibro = listaLibros.getFirst();

* **Eliminar (y obtener) el primer elemento**

Libro primerLibro = listaLibros.removeFirst();

* **Eliminar (y obtener) el último elemento**

Libro ultimoLibro = listaLibros.removeLast();

* **Eliminar (y obtener) un elemento**

Libro unLibro = listaLibros.remove(3);

* **Eliminar todos los elementos**

listaLibros.clear();

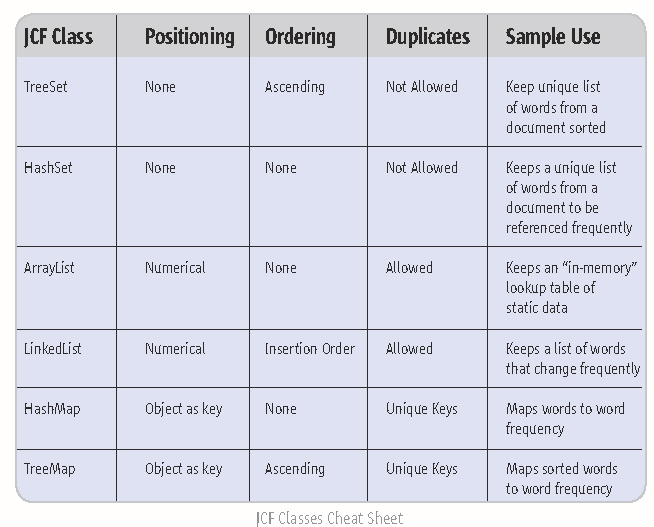
* **Obtener el número de elementos de la lista**

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Esta lista tiene " + listaLibros.size() + " libros");

* **Obtener un array (estático) con todos los elementos de la lista**

Libro[] arrayLibros = listaLibros.toArray();

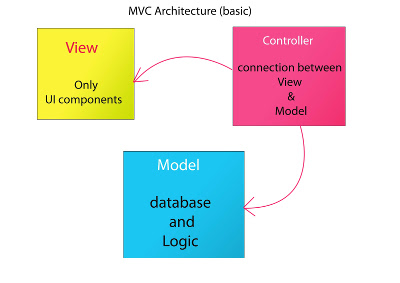
## Comparativa de la JFC (Java Collection Framework)

[](https://datos.codeandcoke.com/_detail/apuntes:comparison.jpg?id=apuntes%3Aficheros)

*JCF*: [Java Collection Framework](https://docs.oracle.com/javase/8/docs/technotes/guides/collections/index.html)

**Patrón Modelo-Vista-Controlador**

El patrón Modelo-Vista-Controlador o MVC (Model-View-Controller en inglés) es un patrón de diseño que busca separar, en una aplicación con GUI, la lógica de la presentación de los datos facilitando así la reutilización de código y el mantenimiento de la aplicación.

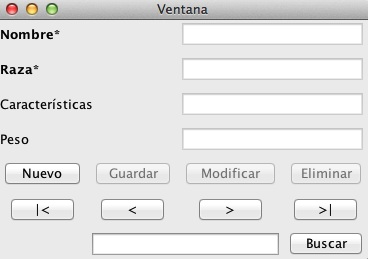
[](https://datos.codeandcoke.com/_detail/apuntes:mvc.jpg?id=apuntes%3Aficheros)

**Figure 1:** Model-View-Controller

**Ejemplo de aplicación MVC**

A continuación se muestra el código completo de una pequeña aplicación con GUI de gestión con las operaciones más básicas (Alta, Modificar, Baja, Búsqueda) gestionada mediante colecciones Java.

Según el patrón MVC, la aplicación quedará separada en tres partes: *Model*, *View* y *Controller*. A esas 3 partes habrá que añadir las clases *POJOs (Plain Old Java Object)* que son las clases que representan a los objetos o elementos que se gestionan en la aplicación. Contendrán atributos, constructores, getters, setters y los métodos que proceda implementar para cada clase. En este caso hay una única clase *POJO* que es Animal.java puesto que la aplicación permite gestionar una base de datos de animales, en este caso mediante colecciones Java de forma que queda como ejercicio muy interesante transformar esta aplicación en una que gestione la información con Bases de Datos (MySQL ó PostgreSQL). Será entonces cuando se aprecie la separación Model-View-Controller y la reutilización real de código entre ambas versiones de esta aplicación.

[](https://datos.codeandcoke.com/_detail/apuntes:gestion_animales.png?id=apuntes%3Aficheros)

[Animal.java](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=67)

/\*\*

\* Clase que representa a un Animal

\*/

public class Animal {

private [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nombre;

private [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) raza;

private [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) caracteristicas;

private float peso;

*// Constructores, getters y setters*

*// Otros métodos (cuando proceda)*

}

La clase Animal representa a los animales que vamos a gestionar y contendrá tantos atributos como características queramos almacenar del mismo. Además haremos los correspondientes constructores, *getters* y *setters*. Si procede, podemos añadir métodos que realicen operaciones sobre la clase.

[Animal.java](https://bitbucket.org/sfaci/java-ficheros/src/9625f045ba0e455e8f879a81b9535a83445ca201/GestionAnimales/src/org/sfaci/gestionanimales/base/Animal.java?at=master&fileviewer=file-view-default) (Todo el código)

[Ventana.java](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=68)

/\*\*

\* Clase que contiene la GUI de la Aplicación

\*/

public class Ventana {

private [JPanel](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jpanel) panel;

[JTextField](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jtextfield) tfNombre;

[JTextField](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jtextfield) tfRaza;

*// Resto de controles swing*

. . .

public Ventana() {

[JFrame](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jframe) frame = new [JFrame](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jframe)("Ventana");

frame.setContentPane(panel);

frame.setDefaultCloseOperation([JFrame](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jframe).EXIT\_ON\_CLOSE);

frame.pack();

frame.setLocationRelativeTo(**null**);

frame.setVisible(**true**);

}

}

La clase Ventana contendrá exclusivamente el diseño de la ventana y en el constructor la inicialización de la misma según convenga en cada caso. Ni siquiera los *Listeners* aparecerán en esta ventana. Por comodidad, no utilizaremos ningún modificador de accesibilidad para los componentes de la ventana, puesto que deberemos poder acceder a ellos desde el *Controller*. Otra manera será proporcionar *getters* para todos ellos y así podremos declararlos como privados o lo que mejor convenga.

[Ventana.java](https://bitbucket.org/sfaci/java-ficheros/src/9625f045ba0e455e8f879a81b9535a83445ca201/GestionAnimales/src/org/sfaci/gestionanimales/gui/Ventana.java?at=master&fileviewer=file-view-default) (Todo el código)

[VentanaController.java](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=69)

/\*\*

\* Controlador para la ventana

\* Recibe los inputs del usuario desde la ventana (View)

\* y se comunica con el Modelo.

\*/

public class VentanaController implements [ActionListener](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+actionlistener), [KeyListener](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+keylistener) {

private VentanaModel model;

private Ventana view;

private int posicion;

public VentanaController(VentanaModel model, Ventana view) {

this.model = model;

this.view = view;

anadirActionListener(this);

anadirKeyListener(this);

posicion = 0;

}

@Override

public void actionPerformed([ActionEvent](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+actionevent) event) {

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) actionCommand = event.getActionCommand();

Animal animal = **null**;

switch (actionCommand) {

case "Nuevo":

view.tfNombre.setText("");

view.tfNombre.setEditable(**true**);

view.tfCaracteristicas.setText("");

view.tfCaracteristicas.setEditable(**true**);

*// Limpiar/Resetear resto de componentes*

. . .

. . .

view.btGuardar.setEnabled(**true**);

break;

case "Guardar":

if (view.tfNombre.getText().equals("")) {

[Util](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+util).mensajeError("El nombre es un campo obligatorio", "Nuevo Animal");

return;

}

animal = new Animal();

animal.setNombre(view.tfNombre.getText());

animal.setRaza(view.tfRaza.getText());

animal.setCaracteristicas(view.tfCaracteristicas.getText());

animal.setPeso([Float](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+float).parseFloat(view.tfPeso.getText()));

model.guardar(animal);

view.btGuardar.setEnabled(**false**);

break;

case "Modificar":

animal = new Animal();

animal.setNombre(view.tfNombre.getText());

animal.setRaza(view.tfRaza.getText());

animal.setCaracteristicas(view.tfCaracteristicas.getText());

animal.setPeso([Float](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+float).parseFloat(view.tfPeso.getText()));

model.modificar(animal);

break;

case "Cancelar":

view.tfNombre.setEditable(**false**);

view.tfCaracteristicas.setEditable(**false**);

. . .

animal = model.getActual();

cargar(animal);

view.btGuardar.setEnabled(**false**);

break;

case "Eliminar":

if ([JOptionPane](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+joptionpane).showConfirmDialog(**null**, "¿Está seguro?", "Eliminar",

[JOptionPane](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+joptionpane).YES\_NO\_OPTION) == [JOptionPane](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+joptionpane).NO\_OPTION)

return;

model.eliminar();

animal = model.getActual();

cargar(animal);

break;

case "Buscar":

animal = model.buscar(view.tfBusqueda.getText());

if (animal == **null**) {

[Util](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+util).mensajeInformacion("No se ha encontrado ningún animal con ese nombre",

"Buscar");

return;

}

cargar(animal);

break;

. . .

. . .

default:

break;

}

}

. . .

. . .

@Override

public void keyReleased([KeyEvent](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+keyevent) e) {

if (e.getSource() == view.tfBusqueda) {

if (e.getKeyCode() == [KeyEvent](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+keyevent).VK\_ENTER) {

view.btBuscar.doClick();

}

}

}

/\*\*

\* Carga los datos de un animal en la vista

\* @param animal

\*/

private void cargar(Animal animal) {

if (animal == **null**)

return;

view.tfNombre.setText(animal.getNombre());

view.tfCaracteristicas.setText(animal.getCaracteristicas());

. . .

. . .

}

. . .

/\*\*

\* Añade el Listener a todos los controles de la View

\*/

private void anadirActionListener([ActionListener](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+actionlistener) listener) {

view.btNuevo.addActionListener(listener);

view.btGuardar.addActionListener(listener);

. . .

. . .

}

}

La clase VentanaController contiene el *Controller* de la ventana, que hará de intermediario entre la capa *View* que es la GUI de la aplicación y el *Model* que contiene toda la lógica y el acceso a los datos. Además, se encarga de gestionar los eventos de cada componente de la GUI (implementando los *Listeners* necesarios).

También se incluyen en este componente todas las validaciones de datos relacionadas con el *input* del usuario y la visualización de los mensajes de errores que correspondan.

[VentanaController.java](https://bitbucket.org/sfaci/java-ficheros/src/9625f045ba0e455e8f879a81b9535a83445ca201/GestionAnimales/src/org/sfaci/gestionanimales/gui/VentanaController.java?at=master) (Todo el código)

[VentanaModel.java](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=70)

/\*\*

\* Modelo para la ventana

\*/

public class VentanaModel {

private ArrayList<Animal> listaAnimales;

private int posicion;

public VentanaModel() {

listaAnimales = new ArrayList<>();

posicion = 0;

}

/\*\*

\* Guarda un animal en la lista

\* @param animal

\*/

public void guardar(Animal animal) {

listaAnimales.add(animal);

posicion++;

}

/\*\*

\* Modifica los datos del animal actual

\* @param animalModificado

\*/

public void modificar(Animal animalModificado) {

Animal animal = listaAnimales.get(posicion);

animal.setNombre(animalModificado.getNombre());

animal.setCaracteristicas(animalModificado.getCaracteristicas());

animal.setRaza(animalModificado.getRaza());

animal.setPeso(animalModificado.getPeso());

}

/\*\*

\* Elimina el animal actual

\*/

public void eliminar() {

listaAnimales.remove(posicion);

}

public Animal getActual() {

return listaAnimales.get(posicion);

}

/\*\*

\* Busca un animal en la lista

\* @param nombre El nombre del animal

\* @return El animal o null si no se ha encontrado nada

\*/

public Animal buscar([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) nombre) {

for (Animal animal : listaAnimales) {

if (animal.getNombre().equals(nombre)) {

return animal;

}

}

return **null**;

}

/\*\*

\* Obtiene el animal que está en primera posición en la lista

\* @return

\*/

public Animal getPrimero() {

posicion = 0;

return listaAnimales.get(posicion);

}

/\*\*

\* Obtiene el animal que está en la posición anterior a la actual

\* @return

\*/

public Animal getAnterior() {

if (posicion == 0)

return **null**;

posicion--;

return listaAnimales.get(posicion);

}

/\*\*

\* Obtiene el animal que está en la posición siguiente a la actual

\* @return

\*/

public Animal getSiguiente() {

if (posicion == listaAnimales.size() - 1)

return **null**;

posicion++;

return listaAnimales.get(posicion);

}

/\*\*

\* Obtiene el animal que está en la última posición de la lista

\* @return

\*/

public Animal getUltimo() {

posicion = listaAnimales.size() - 1;

return listaAnimales.get(posicion);

}

}

La clase VentanaModel, puesto que es el *Model*, contendrá todos los métodos que proporcionan el acceso a los datos y la lógica de la aplicación. En este caso contiene todos los métodos que permite obtener la información de los animales, los resultados de las búsquedas y también se llevan a cabo las operaciones de inserción, modificación y eliminación.

En este caso todas las operaciones se realizan sobre una colección ArrayList. Si quisiéramos hacer que la aplicación utilizara una Base de Datos como motor de almacenamiento sólo tendríamos que modificar este componente de la aplicación haciendo que los mismos métodos (incluso respetando su declaración: nombre, tipo devuelto y parámetros) utilicen una Base de Datos para realizar todas las operaciones. No sería necesario realizar ninguna modificación en el resto de la aplicación.

[VentanaModel](https://bitbucket.org/sfaci/java-ficheros/src/9625f045ba0e455e8f879a81b9535a83445ca201/GestionAnimales/src/org/sfaci/gestionanimales/gui/VentanaModel.java?at=master) (Todo el código)

[Aplicacion.java](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=71)

/\*\*

\* Aplicación para la gestión de animales utilizando el patrón MVC

\*

\* Clase principal, que solamente lanza la aplicación

\*/

public class Aplicacion {

public static void main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) args[]) {

Ventana view = new Ventana();

VentanaModel model = new VentanaModel();

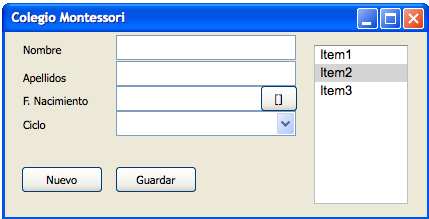
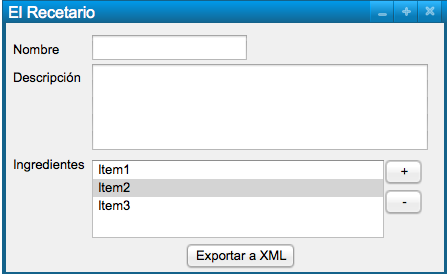
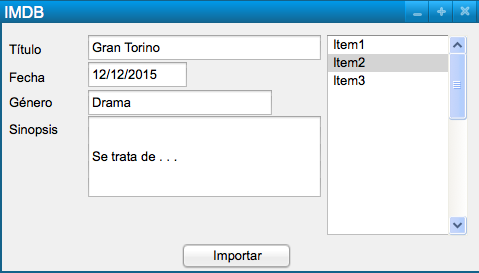
VentanaController controller = new VentanaController(model, view);

}

}

En la clase Aplicacion, que contiene el método main que lanzará la aplicación, simplemente tenemos que instanciar los tres componentes de nuestra aplicación (*Model*, *View* y *Controller*) y “conectarlos” mediante el último.

**Ejercicios**

1. Escribe un programa capaz de registrar información sobre los alumnos de un Colegio (nombre, apellidos, fecha de nacimiento y ciclo que estudian) y listarla en un JList. La información que se muestra en la lista se mantendrá actualizada en todo momento. Diseña la aplicación según el patrón de diseño MVC (Model-View-Controller)[](https://datos.codeandcoke.com/_detail/apuntes:colegio.png?id=apuntes%3Aficheros)
2. Añade al programa anterior los botones necesarios para poder “navegar” (ir al primero, anterior, siguiente y último) por los registros de la lista
3. Añadir un botón al primer ejercicio para que el usuario guarde la información de la lista en un fichero. El usuario tendrá que escoger cada vez el fichero donde quiere almacenar la información
4. Crea una aplicación que sea capaz de exportar a XML la información de una receta (nombre, descripción, ingredientes)[](https://datos.codeandcoke.com/_detail/apuntes:recetario.png?id=apuntes%3Aficheros)
5. Crea una aplicación que sea capaz de importar un fichero XML con información de un catálogo de películas (titulo, fecha, genero, sinopsis y actores principales). Diseña la GUI para que se pueda visualizar toda la información de cada una de las películas importadas[](https://datos.codeandcoke.com/_detail/apuntes:imdb.png?id=apuntes%3Aficheros)
6. Crea una aplicación que sirva de Bloc de Notas con las funciones de *Nuevo*, *Guardar* y *Guardar como*. Diseña la aplicación según el patrón de diseño MVC (Model-View-Controller)

**Ejercicios Examen**

1. Realiza una aplicación que almacene en un fichero una lista de objetos Ordenador (modelo, marca, precio y fecha de compra)
2. Implementa una aplicación que contenga un JComboBox que sea capaz de listar el contenido de una lista de objetos Ordenador (modelo, marca, precio y fecha de compra), mostrando para cada uno de ellos el nombre y su precio
3. Escribe una aplicación con interfaz gráfico que sea capaz de almacenar una lista de objetos Persona (nombre, apellidos, fecha de nacimiento y salario) en un fichero. La aplicación tendrá que funcionar siempre mostrando errores cuando se introduzcan valores incorrectos. El usuario tendrá que ser capaz de:
   1. Registrar personas
   2. Exportar toda la información almacenada a un fichero XML
   3. Listar las personas registradas
4. Escribe una aplicación que añada cadenas de texto a un JList y que contenga un botón que almacene todas ellas a un fichero
5. Implementa una aplicación que contenga un JList que sea capaz de listar el contenido de una lista de objetos Ordenador (con modelo, marca, precio y fecha de compra) mostrando para cada uno de ellos el nombre y su precio
6. Escribe una aplicación que contenga un JComboBox que liste cadenas de texto y que contenga un botón que almacene todas ellas en un fichero
7. Realiza una aplicación que almacene en un fichero la información de un objeto Persona (nombre, apellidos, salario y fecha de nacimiento)
8. Realiza una aplicación para registrar un número indeterminado de personas (almacenando de cada una el nombre, apellidos, salario y fecha de nacimiento), permitiendo también la opción de listarlas mostrando nombre y fecha de nacimiento.
9. Realiza una aplicación que permita añadir cadenas de texto a un JList y que permita almacenarlas a un fichero que el usuario escogerá
10. Realiza una aplicación que almacene en un fichero una serie de datos introducidos por el usuario (nombre, apellidos, fecha de nacimiento, dirección postal y email)
11. Realiza una aplicación que contenga un JComboBox que sea capaz de listar una lista de cadenas de texto que están almacenadas en un fichero
12. Escribe una aplicación que importe un fichero XML cuyo formato se indica a continuación:

[enemigos.xml](https://datos.codeandcoke.com/_export/code/apuntes:ficheros?codeblock=72)

<?xml version="1.0" encoding="UTF-8" standalone="no">

<xml>

<pantalla>

<enemigo>

<nombre>superman</nombre>

<ataque>15</ataque>

<vida>100</vida>

</enemigo>

<enemigo>

<nombre>batman</nombre>

<ataque>150</ataque>

<vida>200</vida>

</enemigo>

<enemigo>

<nombre>spiderman</nombre>

<ataque>5</ataque>

<vida>10</vida>

</enemigo>

</pantall>

</xml>

**Proyectos de ejemplo**

Todos los proyectos de ejemplo de esta parte están en el [repositorio java-ficheros](http://www.github.com/codeandcoke/java-ficheros) de GitHub.

Los proyectos que se vayan haciendo en clase estarán disponibles en el [repositorio datos-ejercicios](http://www.github.com/codeandcoke/datos-ejercicios), también en GitHub.

Para manejaros con Git recordad que tenéis una serie de videotutoriales en la sección [Referencias](https://datos.codeandcoke.com/extra:referencias)

**Práctica 1.1**

**Objetivos**

Desarrollar una aplicación que almacene datos en un fichero

**Enunciado**

Se debe implementar una aplicación que gestione información almacenándola y recuperándola de fichero. Se pensará en un supuesto real en el que exista un tipo de objeto y se creará la aplicación para gestionarlo.

**Requisitos (1 pto cada uno)**

* Se debe gestionar información de un tipo de objeto con, al menos, 5 atributos. Deben aparecer, al menos, datos de tipo cadena, número (entero y decimal) e imágenes.
* De cada objeto el usuario podrá dar de alta, modificar y eliminar
* La aplicación será capaz de almacenar toda la información en disco en una ubicación fija de forma transparente para el usuario. La carga de los datos se realizará durante la carga de la aplicación
* Se deberá comentar el código
* La aplicación contará con un listado para cada tipo de objeto de forma que el usuario pueda acceder a cualquiera de ellos directamente para ver su información

**Otras funcionalidades (1 pto cada una)**

* La aplicación contará con una opción de búsqueda avanzada desde la que se podrá buscar algún objeto
* La aplicación dispondrá de una opción de *guardar como* que permitirá almacenar los datos en una ubicación alternativa
* Añadir la opción de que sea el usuario quién decida cuando realizar el guardado de los datos, en lugar de hacerlo de forma transparente
* Permitir al usuario cambiar, desde la aplicación, la ruta fija donde se almacenan los ficheros
* Añadir una opción a la aplicación que permita eliminar todos los datos del programa
* Activar/Desactivar los controles del interfaz de forma que no se permita al usuario utilizar aquellos que no deba usar en cada momento
* Añadir una opción al usuario que permita recuperar el último elemento borrado

ANEXOS.

Cuando se dice que una estructura de datos, como un ArrayList, "no está sincronizada", significa que no es segura para el uso concurrente por múltiples hilos sin una gestión adicional. En otras palabras, si varios hilos intentan modificar o acceder a un ArrayList al mismo tiempo, puede producirse un comportamiento inesperado o erróneo, como condiciones de carrera, inconsistencias en los datos o incluso excepciones.

Por ejemplo, si un hilo está añadiendo un elemento al ArrayList mientras otro hilo está iterando sobre él, el hilo que itera podría experimentar un error, como una ConcurrentModificationException, porque la estructura de datos ha cambiado mientras estaba siendo recorrida.

Para hacer que una estructura de datos como ArrayList sea segura para su uso en un entorno con múltiples hilos, se pueden usar varias estrategias, como:

1. **Sincronización manual**: Usar bloques sincronizados (synchronized) para controlar el acceso al ArrayList de manera que solo un hilo pueda modificarlo o acceder a él en un momento dado.
2. **Colecciones sincronizadas**: Utilizar versiones sincronizadas de las colecciones proporcionadas por Java, como Collections.synchronizedList(new ArrayList<>()), que envuelven el ArrayList en una estructura sincronizada automáticamente.
3. **Clases concurrentes**: Usar estructuras de datos diseñadas para la concurrencia, como CopyOnWriteArrayList, que manejan la concurrencia de manera más eficiente y segura que simplemente sincronizar manualmente el acceso.

En resumen, una estructura de datos no sincronizada requiere que el programador gestione manualmente la sincronización cuando se trabaja en un entorno de múltiples hilos para evitar problemas de concurrencia.