

## Tema 8: Recursividad

### 1. API del lenguaje java

Cada versión del lenguaje Java tiene una documentación bastante exhaustiva con todas las clases de que dispone el JDK. Estas clases y sus métodos forman lo que se llama el API del lenguaje (Application Programming Interface).

En el caso de la versión que usamos en clase, la 11, podemos acceder a la documentación de la API escribiendo en Google "Java api 11" o clicando directamente en este [link](#).

Nos encontraremos con este contenido:

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Java SE 11 & JDK 11

ALL CLASSES SEARCH: Search

This document is divided into two sections:

**Java SE**  
The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

**JDK**  
The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

Module	Description
<code>java.base</code>	Defines the foundational APIs of the Java SE Platform.
<code>java.compiler</code>	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
<code>java.datatransfer</code>	Defines the API for transferring data between and within applications.

Si hacemos clic en el enlace "ALL CLASSES" de la parte superior izquierda, accederemos a un listado con todas las clases de Java ordenadas alfabéticamente.

All Classes

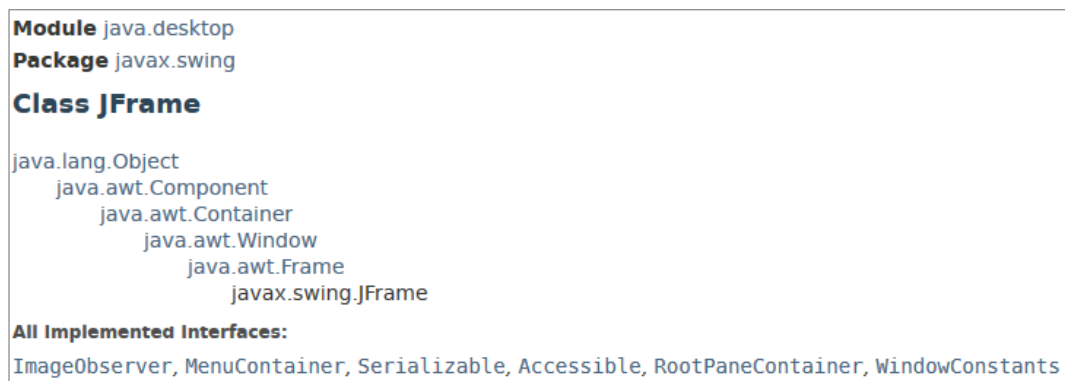
- AboutEvent
- AboutHandler
- AbsentInformationException
- AbstractAction
- AbstractAnnotationValueVisitor6
- AbstractAnnotationValueVisitor7
- AbstractAnnotationValueVisitor8
- AbstractAnnotationValueVisitor9
- AbstractBorder
- AbstractButton
- AbstractCellEditor

Si clicamos en un nombre de clase, se nos abrirá la página con la documentación para esa clase. Por ejemplo, para la clase JFrame:

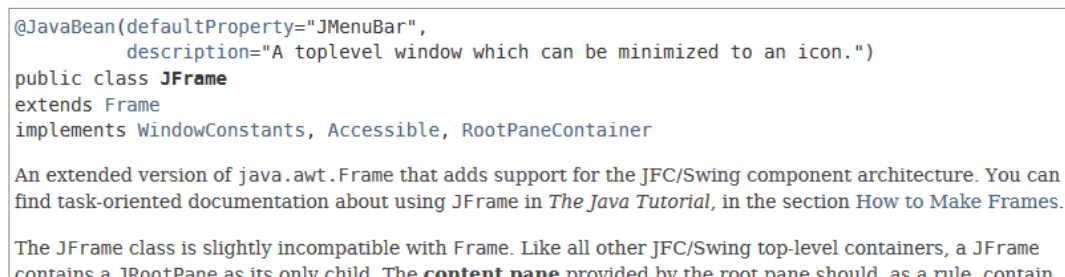


En esta página podremos ver:

- La jerarquía de herencia que lleva desde la clase Object a la clase JFrame, y las interfaces que implementa:



- Una descripción general del propósito de la clase:



- Un listado de las interfaces y clases internas en las clases de las que hereda JFrame:

<b>Nested Class Summary</b>		
<b>Nested Classes</b>		
Modifier and Type	Class	Description
protected class	<b>JFrame.AccessibleJFrame</b>	This class implements accessibility support for the JFrame class.
<b>Nested classes/interfaces declared in class java.awt.Frame</b>		
Frame.AccessibleAWTFrame		
<b>Nested classes/interfaces declared in class java.awt.Window</b>		

- Un resumen de los atributos de la clase:

<b>Field Summary</b>		
<b>Fields</b>		
Modifier and Type	Field	Description
protected <b>AccessibleContext</b>	<b>accessibleContext</b>	The accessible context property.
protected <b>JRootPane</b>	<b>rootPane</b>	The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
protected boolean	<b>rootPaneCheckingEnabled</b>	If true then calls to add and setLayout will be forwarded to the contentPane.

- Un listado de atributos que hereda de otras clases:

<b>Fields declared in class java.awt.Frame</b>	
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR	
<b>Fields declared in class java.awt.Component</b>	
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT	

- Un resumen de los constructores de la clase:

<b>Constructor Summary</b>	
<b>Constructors</b>	
<b>Constructor</b>	<b>Description</b>
<code>JFrame()</code>	Constructs a new frame that is initially invisible.
<code>JFrame(GraphicsConfiguration gc)</code>	Creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
<code>JFrame(String title)</code>	Creates a new, initially invisible Frame with the specified title.
<code>JFrame(String title, GraphicsConfiguration gc)</code>	Creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

- Un resumen de los métodos de la clase:

<b>Method Summary</b>			
<b>All Methods</b>	<b>Static Methods</b>	<b>Instance Methods</b>	<b>Concrete Methods</b>
<b>Modifier and Type</b>	<b>Method</b>	<b>Description</b>	
protected void	<code>addImpl(Component comp, Object constraints, int index)</code>	Adds the specified child Component.	
protected <code>JRootPane</code>	<code>createRootPane()</code>	Called by the constructor methods to create the default rootPane.	
protected void	<code>frameInit()</code>	Called by the constructors to init the JFrame properly.	
<code>AccessibleContext</code>	<code>getAccessibleContext()</code>	Gets the AccessibleContext associated with this JFrame.	

- Una lista de métodos heredados de otras clases:

<b>Methods declared in class java.awt.Frame</b>
addNotify, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated
<b>Methods declared in class java.awt.Window</b>
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle,

- Un listado de los atributos de la clase detallados:

**Field Detail**

**rootPane**

```
protected JRootPane rootPane
```

The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.

**See Also:**  
JRootPane, RootPaneContainer

- Un listado de los constructores, con detalles:

**Constructor Detail**

**JFrame**

```
public JFrame()  
    throws HeadlessException
```

Constructs a new frame that is initially invisible.

This constructor sets the component's locale property to the value returned by `JComponent.getDefaultLocale`.

**Throws:**  
HeadlessException - if `GraphicsEnvironment.isHeadless()` returns true.

**See Also:**  
`GraphicsEnvironment.isHeadless()`, `Component.setSize(int, int)`, `Component.setVisible(boolean)`, `JComponent.getDefaultLocale()`

- Un listado de los métodos, con detalles:

**Method Detail**

**frameInit**

```
protected void frameInit()
```

Called by the constructors to init the JFrame properly.

## 2. Documentación de programas en java

Ya conocemos la utilidad de escribir comentarios en un programa:

- Facilita la comprensión y el uso de nuestro código por parte de otras personas. Hay que pensar que habitualmente trabajaremos en una organización, como parte de un equipo, y otros programadores tendrán que entender lo que nosotros hayamos programado.
- También nos facilita la comprensión a nosotros mismos, cuando llevemos tiempo sin haber trabajado con la aplicación y tengamos que hacerle alguna modificación.
- Permite detectar bugs más rápido en nuestro programa, al permitir entender fácilmente que hacen las distintas partes de nuestro programa.

Hasta ahora hemos usado dos tipos de comentarios en Java:

- Los de una sola línea, que empiezan por `//`.
- Los de varias líneas, que van comprendidos entre `/*` y `*/`.

Además de éstos, que son para usar dentro de métodos, Java tiene otra modalidad, que sirve para generar a partir de ellos documentación del programa. Estos comentarios irán siempre en un bloque de este tipo:

```
/**  
 */
```

Para crear la documentación de un programa, a partir de los comentarios en su código fuente, tenemos la herramienta **Javadoc**, que viene incluida en el JDK.

Cuando ejecutamos el programa *javadoc*, estos comentarios se insertarán en el HTML de un sitio web local generado en el momento. Si abrimos esta web, veremos que tiene exactamente el mismo aspecto que la documentación del API de Java.

Los IDEs como Eclipse entienden perfectamente los comentarios tipo javadoc, lo que les permite mostrar información contextual muy útil a la hora de programar.

## 2.1 Comentarios de clase

Antes de la instrucción "public class NombreClase" colocaremos siempre un comentario donde habrá información general de la clase. El formato será:

- Descripción breve de la clase.
- Información más extensa de la clase. Cada párrafo que queramos añadir irá precedido de una línea con una etiqueta `<p>`.

Podemos añadir información opcional a continuación:

- Autor de la clase, precedido por la etiqueta `@author`.
- Versión de la clase, precedida por la etiqueta `@version`. Un formato recomendado es escribir la versión, una coma y la fecha en formato "día mes año", en inglés.
- Enlaces a la documentación de otras clases relacionadas, para facilitar al lector conseguir la información. Irán precedidos de la etiqueta `@see`.

Ejemplo de comentario para una clase llamada Puntuaciones:

```
/**
 * La clase Puntuaciones es la clase principal del programa Hall of Fame.
 *
 * <p>
 * Hall of Fame permite registrar una lista de jugadores y puntuaciones que
 * queda guardada en un archivo de disco.
 *
 * @see Jugador
 * @author Carlos Sogorb
 * @version 0.1, 8 May 2020
 */
public class Puntuaciones extends JFrame {}
```

## 2.2 Comentarios de atributo

Antes de la declaración de cada atributo podemos añadir un comentario siguiendo este esquema:

- Descripción breve del atributo.
- Información más extensa del atributo, si es necesaria. Cada párrafo que queramos añadir irá precedido de una línea con una etiqueta `<p>`.

Ejemplo de comentario de un atributo de una clase:

```
/**  
 * Nombre del jugador  
 * <p>  
 * Éste es el nombre que se mostrará en la lista de jugadores y puntuaciones  
 */  
private String nombre;
```

## 2.3 Comentarios de método

Antes de la declaración de cada método colocaremos un comentario donde habrá información sobre el método: su función, qué parámetros toma y qué dato devuelve. El formato será:

- Descripción breve del método.
- Información más extensa del método. Cada párrafo que queramos añadir irá precedido de una línea con una etiqueta `<p>`.
- Información de los parámetros que recibe el método. Por cada parámetro aparecerá una línea que empieza por la etiqueta `@param`, seguida del nombre del parámetro y una breve descripción de qué es ese parámetro.
- Una línea que empieza por la etiqueta `@returns` con información sobre el valor devuelto por el método.



Ejemplo de comentario para el constructor de una clase Jugador. Este método toma dos parámetros y no devuelve nada:

```
/**
 * Inicializa un objeto Jugador recién creado.
 *
 * @param nombre el nombre del jugador
 * @param puntos los puntos del jugador
 */
public Jugador(String nombre, int puntos) {}
```

Ejemplo de comentario para un método que no toma ningún parámetro pero devuelve un objeto de tipo Jugador:

```
/**
 * Pide los datos del jugador y devuelve un objeto Jugador.
 *
 * @return el objeto Jugador con los datos introducidos por el usuario o null si
 *         los datos no han sido válidos.
 */
private Jugador leerDatosJugador() {
```

### 3. Generar la documentación de nuestro programa

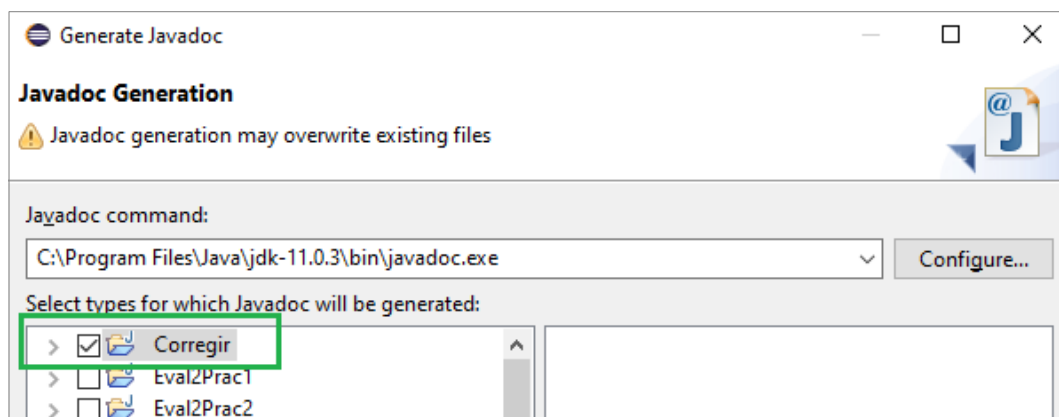
Una vez tenemos nuestro programa comentado con comentarios tipo javadoc, podemos generar la documentación de dos maneras:

- Usando un IDE como Eclipse. De esta manera será el IDE el que se encargará de ejecutar javadoc.
- Usando directamente el programa de consola javadoc.

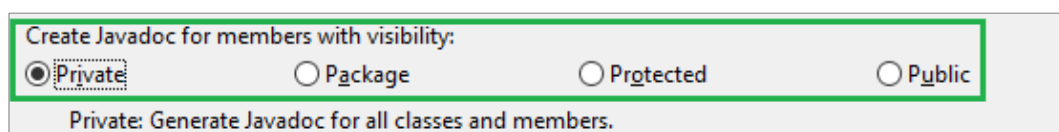
#### 3.1 Generar la documentación con Eclipse

Para generar la documentación a partir de los comentarios que hemos añadido a nuestras clases, haremos clic en el menú *Project -> Generate Javadoc* de Eclipse.

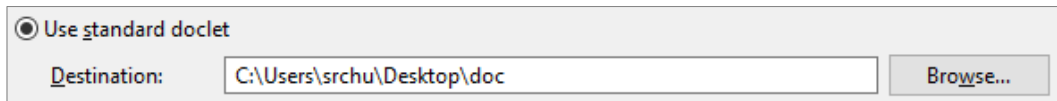
En la ventana que se abre, nos aseguramos de tener marcado el proyecto cuya documentación queremos crear:



Y más abajo, seleccionamos el nivel de visibilidad de los métodos que queramos que aparezcan. Si seleccionamos **Public**, sólo saldrá documentación de los métodos públicos. Si, en cambio, seleccionamos **Private**, se creará documentación de todo:



Después, configuramos la ruta donde queremos que se cree el sitio web. En mi caso, he elegido una carpeta *doc* en mi escritorio:

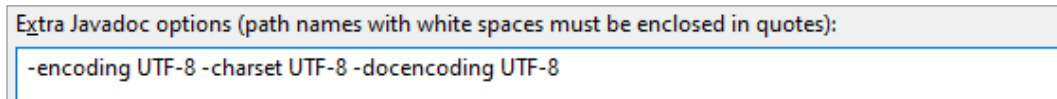


Use standard doclet

Destination: C:\Users\srchu\Desktop\doc Browse...

A continuación pulsamos Next y luego otra vez Next. Entonces nos aparecerá una ventana donde podremos introducir opciones extra para Javadoc. Si queremos escribir comentarios en español, necesitaremos indicarle que utilice codificación UTF-8, con estas instrucciones:

`-encoding UTF-8 -charset UTF-8 -docencoding UTF-8`



Extra Javadoc options (path names with white spaces must be enclosed in quotes):

-encoding UTF-8 -charset UTF-8 -docencoding UTF-8

### 3.2 Generar la documentación desde la terminal de Windows

Para poder usar el programa **javadoc** desde la terminal de Windows necesitaremos que el sistema operativo sepa dónde se encuentra el ejecutable. Para ello editaremos la variable de entorno **Path**, que es una cadena de texto donde están las rutas de las carpetas donde Windows va a buscar cualquier programa que intentemos ejecutar desde la consola.

Para poder ejecutar comandos como `javac`, `java` o `javadoc` tendremos que añadir la ruta a la carpeta donde se encuentran los binarios (ejecutables) de Java. En general la ruta será algo así:

C:\Program Files\Java\jdk-11.0.3\bin

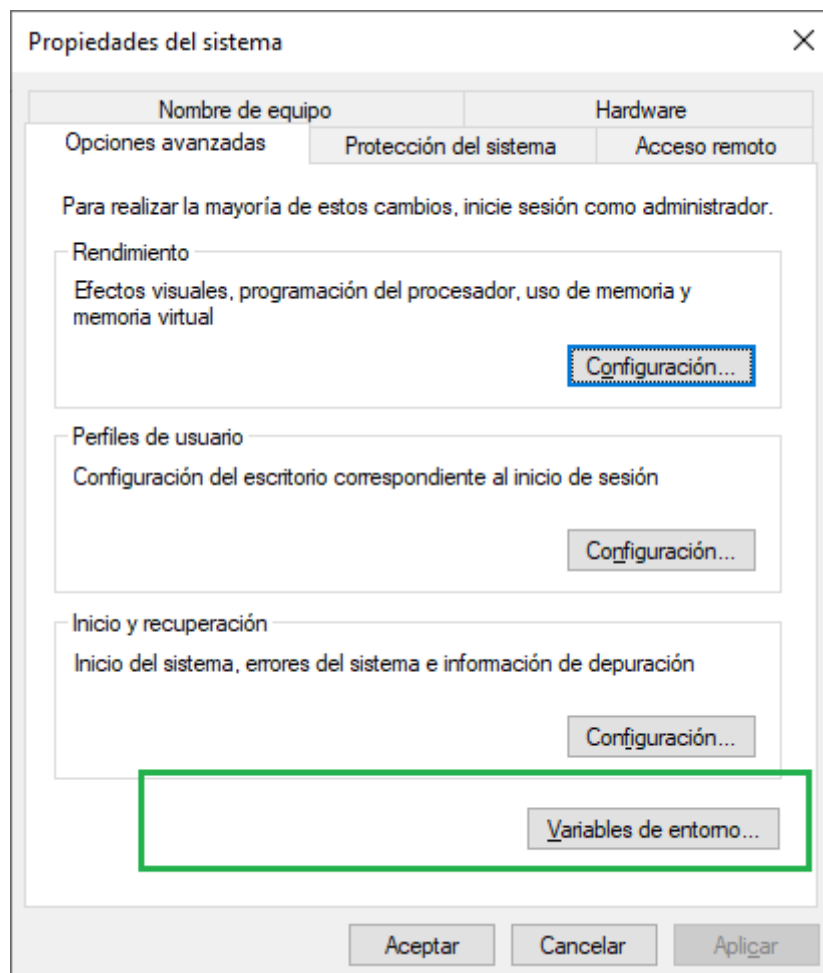
La parte resaltada en amarillo puede cambiar dependiendo de qué versión del JDK tengamos instalado. Lo más práctico es ir hasta la carpeta **bin** con el explorador de archivos y copiar la ruta.

### 3.2.1 Modificar la variable de entorno Path

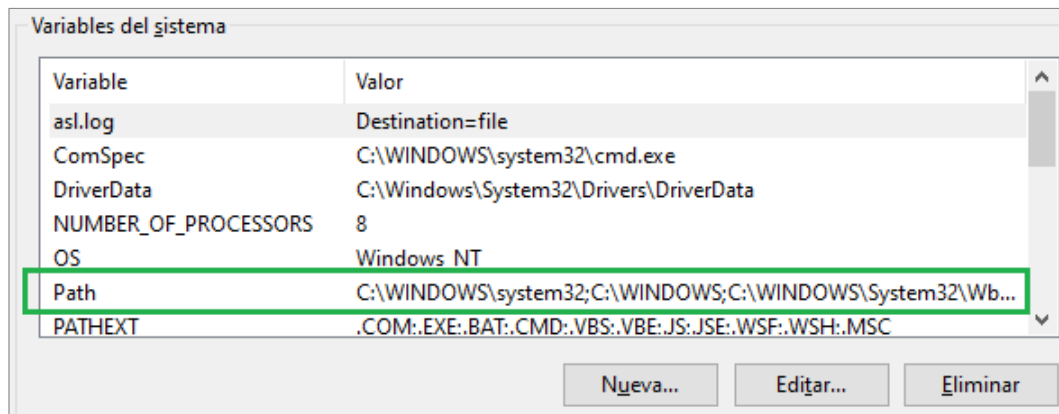
Para modificar la variable Path pulsaremos el botón de Windows y escribiremos "Variables de entorno". Al hacerlo se nos sugerirá la opción siguiente:



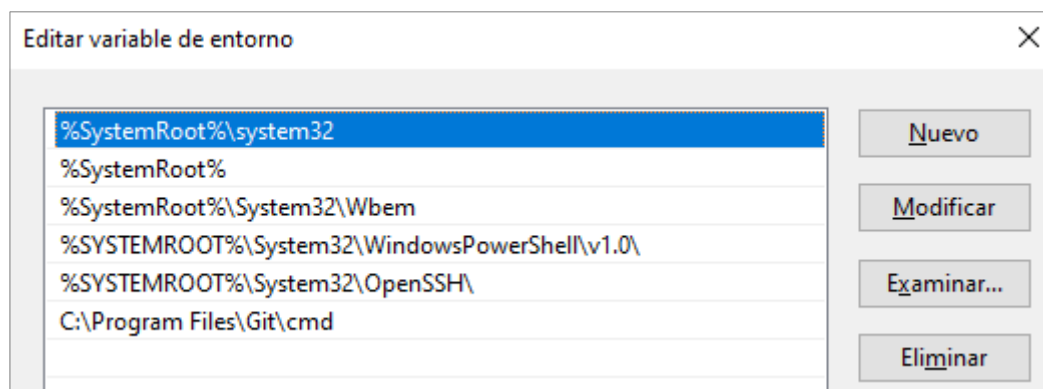
Tras pulsar en "Editar las variables de entorno del sistema" se abrirá la siguiente ventana, y haremos clic en el botón "Variables de entorno":



Tras pulsar el botón, se nos mostrará una ventana con las variables de sistema y de usuario existentes. Nos fijamos en las de sistema, en concreto seleccionamos Path y pulsamos el botón Editar:

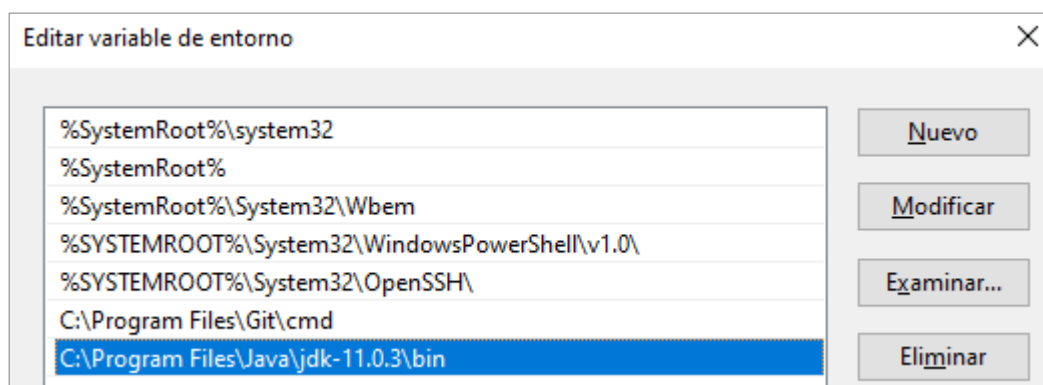


A continuación se mostrará una ventana donde cada fila es la ruta a una carpeta con archivos ejecutables. Para añadir la del javadoc, pulsaremos el botón Nuevo:



Y se habilitará una fila para poner la ruta.

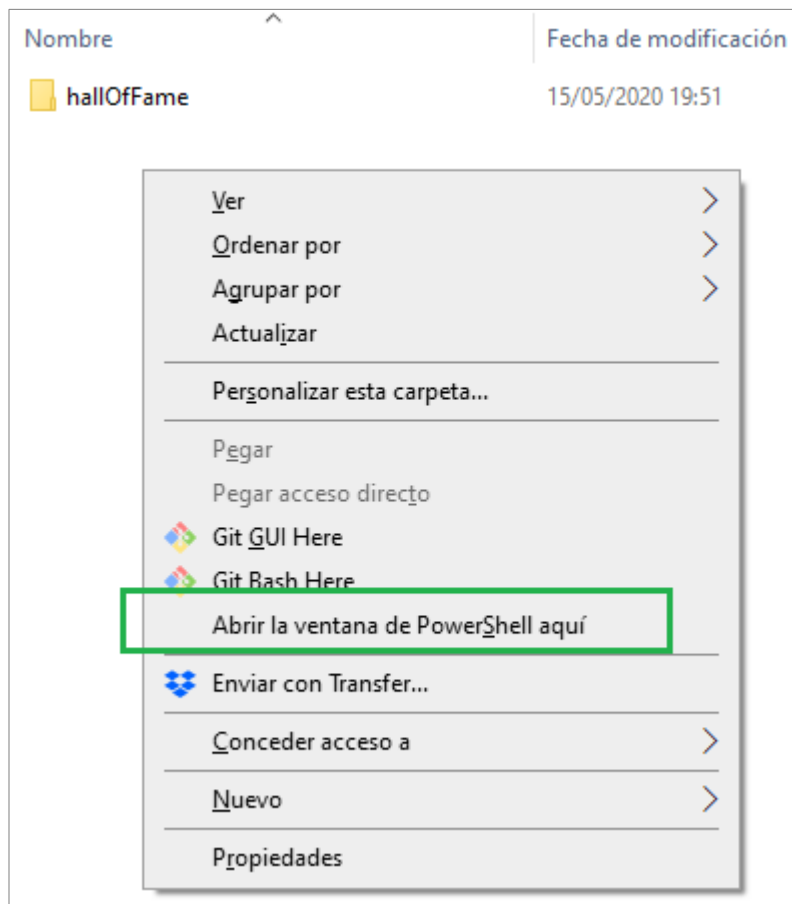
Tras pulsar intro, el cuadro de la variable Path quedará al final más o menos así:



A continuación iremos pulsando Aceptar para cerrar todas las ventanas, y finalmente reiniciaremos el ordenador para que se aplique el cambio.

### 3.2.2 Ejecutar javadoc

Abrimos el explorador de archivos y nos dirigimos a la carpeta src de nuestro proyecto. Pulsamos la tecla SHIFT (mayúsculas) y sin soltarla hacemos clic derecho con el ratón. En el menú contextual, haremos clic en la opción "Abrir la ventana de PowerShell aquí":

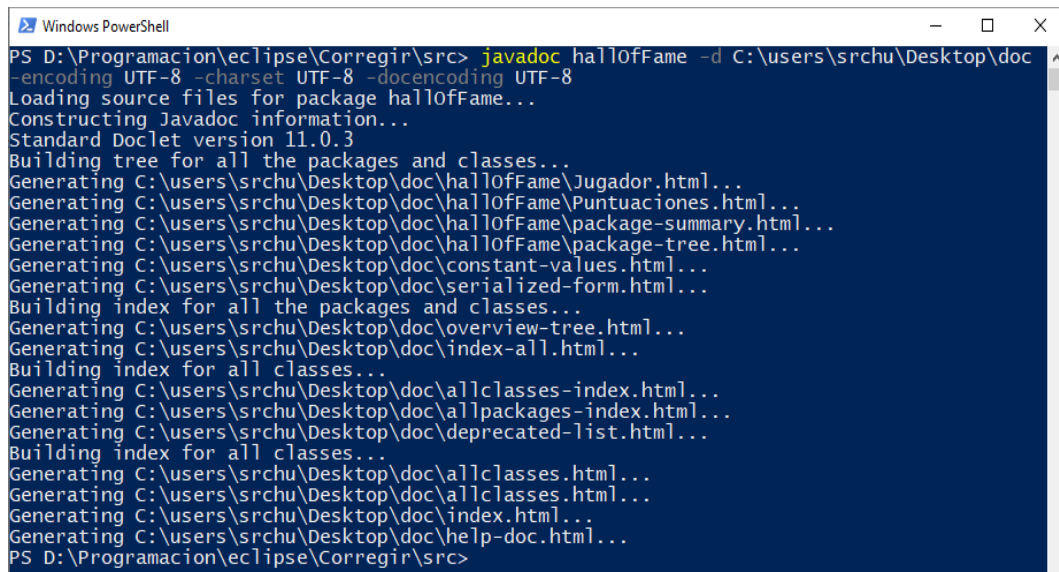


Al hacer esto, se abrirá la terminal PowerShell y podremos ejecutar comandos. Suponiendo que nuestra aplicación está en el paquete hallOfFame, escribiremos la siguiente línea:

```
javadoc hallOfFame -d C:\users\srchu\Desktop\doc -encoding UTF-8 -charset UTF-8  
-docencoding UTF-8
```

Aquí hemos usado el parámetro -d para indicar la ruta en nuestro equipo para la documentación que se creará (he decidido que sea en una carpeta llamada *doc* en mi escritorio), y el resto son parámetros para utilizar codificación utf-8.

Podemos ver el resultado:



```

PS D:\Programacion\eclipse\Corregir\src> javadoc hallOfFame -d C:\users\srchu\Desktop\doc
-encoding UTF-8 -charset UTF-8 -docencoding UTF-8
Loading source files for package hallOfFame...
Constructing Javadoc information...
Standard Doclet version 11.0.3
Building tree for all the packages and classes...
Generating C:\users\srchu\Desktop\doc\hallOfFame\Jugador.html...
Generating C:\users\srchu\Desktop\doc\hallOfFame\Puntuaciones.html...
Generating C:\users\srchu\Desktop\doc\hallOfFame\package-summary.html...
Generating C:\users\srchu\Desktop\doc\hallOfFame\package-tree.html...
Generating C:\users\srchu\Desktop\doc\constant-values.html...
Generating C:\users\srchu\Desktop\doc\serialized-form.html...
Building index for all the packages and classes...
Generating C:\users\srchu\Desktop\doc\overview-tree.html...
Generating C:\users\srchu\Desktop\doc\index-all.html...
Building index for all classes...
Generating C:\users\srchu\Desktop\doc\allclasses-index.html...
Generating C:\users\srchu\Desktop\doc\allpackages-index.html...
Generating C:\users\srchu\Desktop\doc\deprecated-list.html...
Building index for all classes...
Generating C:\users\srchu\Desktop\doc\allclasses.html...
Generating C:\users\srchu\Desktop\doc\allclasses.html...
Generating C:\users\srchu\Desktop\doc\index.html...
Generating C:\users\srchu\Desktop\doc\help-doc.html...
PS D:\Programacion\eclipse\Corregir\src>

```

Aunque aquí hemos usado la PowerShell, se podría hacer exactamente igual con la línea de comandos clásica de Windows.

Por defecto sólo se genera documentación para entidades public. Pero podemos modificar esto usando una de estas opciones: **-public**, **-protected**, **-package**, **-private**. Por ejemplo:

```
javadoc hallOfFame -d C:\users\srchu\Desktop\doc -encoding UTF-8 -charset UTF-8
-docencoding UTF-8 -private
```

Por último, si queremos que se muestre información del autor de las clases, podemos añadir la opción **-author**:

```
javadoc hallOfFame -d C:\users\srchu\Desktop\doc -encoding UTF-8 -charset UTF-8
-docencoding UTF-8 -private -author
```

## 4. Otras características de los comentarios

Podemos cambiar el aspecto de los comentarios usando ciertas etiquetas.

### 4.1 Mostrar listas en los comentarios

Podemos usar las etiquetas `<ul>`, `<ol>` y `<li>`, de HTML, para mostrar información en forma de lista. Por ejemplo:

```
/**
 * La clase Puntuaciones es la clase principal del programa Hall of Fame.
 *
 * <p>
 * Hall of Fame permite registrar una lista de jugadores y puntuaciones que
 * queda guardada en:
 * <ul>
 * <li>Un archivo de disco</li>
 * <li>Un JList visible en la ventana</li>
 * </ul>
 */
```

Se vería después con este aspecto:

```
public class Puntuaciones
extends javax.swing.JFrame

La clase Puntuaciones es la clase principal del programa Hall of Fame.

Hall of Fame permite registrar una lista de jugadores y puntuaciones que queda guardada en:

    • Un archivo de disco
    • Un JList visible en la ventana
```

### 4.2 Resaltar fragmentos de código

Cuando en un comentario nuestro aparezca un trozo de código deberíamos resaltarlo colocándolo entre las etiquetas `<code>``</code>`. Así se mostrará con otra tipografía cuando se genere la documentación. Por ejemplo:

```
/**
 * Pide los datos del jugador y devuelve un objeto Jugador.
 *
 * @return Jugador con los datos introducidos por el usuario o <code>null</code>
 *         si los datos no han sido válidos.
 */
```



### 4.3 Agregar un link a la documentación de otra clase

Podemos añadir en un comentario un link a la documentación de otra clase del proyecto. Para eso ponemos el nombre de la clase dentro de una etiqueta {@link}. Por ejemplo:

```
/**
 * Pide los datos del jugador y devuelve un objeto {@link Jugador}.
 *
 * @return Jugador con los datos introducidos por el usuario o <code>null</code>
 *         si los datos no han sido válidos.
 */
```

Así se vería después:

**leerDatosJugador**

```
private Jugador leerDatosJugador()
```

Pide los datos del jugador y devuelve un objeto **Jugador**.

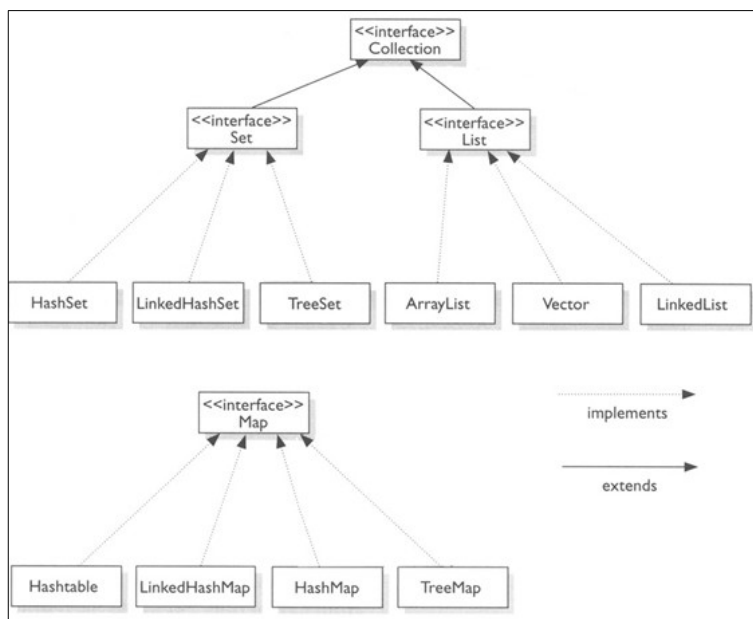
**Returns:**

Jugador con los datos introducidos por el usuario o null si los datos no han sido válidos.

## 5. Las colecciones en Java

Las colecciones en Java representan grupos de objetos. Hasta ahora lo más parecido que conocemos a una colección son los arrays.

Las colecciones derivan de la interface *Collection* y comparten métodos comunes. Todas estas clases se encuentran en el paquete *java.util*.



Las colecciones en Java siempre implementan una de estas tres interfaces:

- La interface *List* define colecciones que son sucesiones de elementos. Estos elementos pueden repetirse.
- La interface *Set* define colecciones cuyos elementos no pueden repetirse.
- La interfaz *Map* asocia claves (por ejemplo, Strings) con valores. Las claves no pueden repetirse.

### 5.1 Listas

Las listas se construyen con clases que implementan la interfaz *List*. Son sucesiones ordenadas en las que los elementos pueden repetirse. Cada elemento tiene un índice o posición.

En la práctica podemos pensar en una lista como un array en el que podemos añadir elementos indefinidamente.

Algunas de las clases que podemos usar para implementar una lista son:

- **ArrayList:** Se basa en un array redimensionable que aumenta su tamaño cuando aumenta la cantidad de elementos que contiene. Es el más utilizado (mejor rendimiento en la mayoría de las ocasiones).
- **LinkedList:** Se basa en una lista doblemente enlazada de los elementos, teniendo cada uno de los elementos un puntero al anterior y al siguiente elemento. Permite inserción y borrado de elementos rápido.
- **Vector:** Muy similar a *ArrayList*, cambia la forma en que se pueden modificar elementos entre varios hilos de ejecución. El acceso a un *Vector* es sincronizado (sólo un hilo de la aplicación puede modificarlo a la vez). En general se recomienda, por razones de rendimiento, usar *ArrayList*.

Ejemplos de creación de listas:

```
List alumnos = new ArrayList();  
List documentos = new LinkedList();  
List nominas = new Vector();
```

Desde Java 5, en que aparecieron los tipos genéricos, se recomienda indicar el tipo de dato que van a almacenar las colecciones en el momento de crearlas:

```
List<String> documentos = new LinkedList<String>();
```

Aunque desde Java 8, podemos dejar que se infiera automáticamente el tipo, quedando la expresión anterior acortada así:

```
List<String> documentos = new LinkedList<>();
```

En general el indicar el tipo de los elementos que va a contener una colección nos permite detectar errores en tiempo de compilación (por ejemplo, al introducir un objeto de otro tipo) y evitar el uso de casts.

## 6. ArrayLists

Como ejemplo, trabajaremos con un ArrayList de objetos String:

```
public static void main(String[] args) {  
    // Declaramos un ArrayList de Strings.  
    ArrayList<String> lista = new ArrayList<>();  
  
    // También lo podríamos haber declarado así:  
    List<String> lista2 = new ArrayList<>();  
}
```

Podemos ver que en ningún momento le asignamos un tamaño al Arraylist.

### 6.1 Añadir elementos a la lista

Dado un Arraylist de objetos de clase E disponemos de dos métodos para añadir elementos:

- boolean **add**(E e): Añade un nuevo elemento *e* al final de la lista.
- boolean **add**(int indice, E e): Inserta un nuevo elemento *e* en la posición *indice*. El elemento que estaba en esta posición y los siguientes se moverán una posición para dejar sitio.

Por ejemplo, si a nuestra lista queremos añadirle cuatro elementos y que se coloquen uno tras otro, podemos usar el primer método *add()*:

```
public static void main(String[] args) {  
  
    // Declaramos un ArrayList de Strings.  
    ArrayList<String> lista = new ArrayList<>();  
  
    // Añadimos elementos al final de la lista.  
    lista.add("Eddard Stark");  
  
    lista.add("Catelyn Stark");  
  
    lista.add("Jon Nieve");  
  
    lista.add("Sansa Stark");  
  
    lista.add("Arya Stark");  
  
}
```

Tras ejecutar este código, la lista tendrá este aspecto:

Índice	Elemento
0	"Eddard Stark"
1	"Catelyn Stark"
2	"Jon Nieve"
3	"Sansa Stark"

Si ahora quisiésemos insertar un elemento en una posición concreta, podemos usar la versión del método `add()` que admite dos parámetros. Por ejemplo, insertaremos el personaje "Robb Stark" en la posición 1:

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

    // Declaramos un ArrayList de Strings.
    ArrayList<String> lista = new ArrayList<>();

    // Añadimos elementos al final de la lista.
    lista.add("Eddard Stark");

    lista.add("Catelyn Stark");

    lista.add("Jon Nieve");

    lista.add("Sansa Stark");

    lista.add("Arya Stark");

    // Añadir un elemento en la posición 1 de una lista.
    lista.add(1, "Robb Stark");

}
```

Y la lista quedaría así:

Índice	Elemento
0	"Eddard Stark"
1	"Robb Stark"
2	"Catelyn Stark"
3	"Jon Nieve"
4	"Sansa Stark"
5	"Arya Stark"

## 6.2 Acceder a un elemento de la lista

Podemos acceder a cualquier posición de un `ArrayList` con el método:

- El `get(int indice)`: Devuelve el elemento guardado en la posición *indice*.

Por ejemplo, para acceder al primer elemento de la lista:

```
// Acceder a un elemento de la lista.
String nombre = lista.get(0);
System.out.println("En la posición 0 tenemos: " + nombre);
```

En este caso el valor mostrado sería "Eddard Stark".

### 6.3 Recorrer una lista

Podemos recorrer una lista elemento a elemento usando un bucle normal con el método *get()*, o utilizar la siguiente estructura *for*:

```
for(E e : lista){
    ...
}
```

Con este tipo de bucle *for*, la variable *e*, de la clase E, irá tomando todos los valores contenidos en la lista, desde la posición 0 hasta el final.

Esta estructura de *for* se conoce como "for mejorado". También se le llama "foreach", ya que en algunos lenguajes se escribe así.

Como ejemplo, podemos añadir a nuestro programa un bucle que nos muestre cómo queda la lista después de las inserciones que hemos hecho:

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

    // Declaramos un ArrayList de Strings.
    ArrayList<String> lista = new ArrayList<>();

    // Añadimos elementos al final de la lista.
    lista.add("Eddard Stark");

    lista.add("Catelyn Stark");

    lista.add("Jon Nieve");

    lista.add("Sansa Stark");

    lista.add("Arya Stark");

    // Añadir un elemento en un posición concreta de una lista.
    lista.add(1, "Robb Stark");

    // Mostramos todos los elementos de la lista.
    for (String personaje : lista) {
        System.out.println(personaje);
    }

}
```

La variable *personaje* irá tomando todos los valores contenidos en la lista, y, por tanto, este bucle nos mostrará todos los nombres guardados en ella.

## 6.4 Eliminar objetos de una lista

Cuando queramos eliminar un elemento de nuestra lista, podemos usar los métodos siguientes:

- boolean **remove**(E e): Elimina la primera ocurrencia del elemento *e*. Devuelve true si el elemento que queremos eliminar estaba en la lista, y false en caso contrario.

Lo que hace exactamente este método es recorrer la lista desde el principio. Para cada elemento de la lista comprueba, con el método *equals()*, si es igual al elemento buscado. Si es así, elimina el elemento, desplaza el resto de elementos para que no queden huecos y termina.

Así que si queremos usar este método *remove()* hay que sobrescribir *equals()* de la clase E para que se comporte como nosotros queramos. Por ejemplo, si tuviésemos un ArrayList de objetos Cliente, tendríamos que sobrescribir el método *equals()* de la clase Cliente.

En el caso de que sea una lista de Strings esto no es necesario, ya que su método *equals()* ya nos sirve.

- E **remove**(int indice): Elimina el elemento de la posición indicada. Devuelve el elemento que hemos eliminado de la lista. El resto de elementos se desplaza para que no queden huecos.



Volviendo a nuestro ejemplo, podemos añadir código para eliminar el elemento de la lista que ocupa la posición 2 (Catelyn Stark):

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

    // Declaramos un ArrayList de Strings.
    ArrayList<String> lista = new ArrayList<>();

    // Añadimos elementos al final de la lista.
    lista.add("Eddard Stark");

    lista.add("Catelyn Stark");

    lista.add("Jon Nieve");

    lista.add("Sansa Stark");

    lista.add("Arya Stark");

    // Añadir un elemento en un posición concreta de una lista.
    lista.add(1, "Robb Stark");

    // Mostramos todos los elementos de la lista.
    for (String personaje : lista) {
        System.out.println(personaje);
    }

    // Eliminamos el elemento que ocupa la posición 2 de la lista.
    lista.remove(2);

}
```

Y la lista quedaría así:

Índice	Elemento
0	"Eddard Stark"
1	"Robb Stark"
2	"Jon Nieve"
3	"Sansa Stark"
4	"Arya Stark"

Puede verse que "Jon Nieve", "Sansa Stark" y "Arya Stark" se han desplazado un sitio hacia arriba para rellenar el hueco dejado por Catelyn Stark.

También podemos guardar el objeto eliminado para hacer algo con él más tarde, como en el siguiente ejemplo:

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

    // Declaramos un ArrayList de Strings.
    ArrayList<String> lista = new ArrayList<>();

    // Añadimos elementos al final de la lista.
    lista.add("Eddard Stark");

    lista.add("Catelyn Stark");

    lista.add("Jon Nieve");

    lista.add("Sansa Stark");

    lista.add("Arya Stark");

    // Añadir un elemento en un posición concreta de una lista.
    lista.add(1, "Robb Stark");

    // Mostramos todos los elementos de la lista.
    for (String personaje : lista) {
        System.out.println(personaje);
    }

    // Eliminamos el elemento que ocupa la posición 2 de la lista.
    lista.remove(2);

    // Eliminar un elemento de la lista aprovechando que el método nos lo
    // devuelve.
    String eliminado = lista.remove(1);
    System.out.println("Acabamos de eliminar a " + eliminado);

}
```

Ahora el programa nos dirá que acabamos de eliminar a Robb Stark, y la lista quedará así:

Índice	Elemento
0	"Eddard Stark"
1	"Jon Nieve"
2	"Sansa Stark"
3	"Arya Stark"

Y, por último, podemos eliminar un elemento de la lista pasándolo directamente, sin indicar su posición. Concretamente, eliminaremos a Arya Stark:

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

    // Declaramos un ArrayList de Strings.
    ArrayList<String> lista = new ArrayList<>();

    // Añadimos elementos al final de la lista.
    lista.add("Eddard Stark");

    lista.add("Catelyn Stark");

    lista.add("Jon Nieve");

    lista.add("Sansa Stark");

    lista.add("Arya Stark");

    // Añadir un elemento en un posición concreta de una lista.
    lista.add(1, "Robb Stark");

    // Mostramos todos los elementos de la lista.
    for (String personaje : lista) {
        System.out.println(personaje);
    }

    // Eliminamos el elemento que ocupa la posición 2 de la lista.
    lista.remove(2);

    // Eliminar un elemento de la lista aprovechando que el método nos lo
    // devuelve. Eliminamos a Robb Stark.
    String eliminado = lista.remove(1);
    System.out.println("Acabamos de eliminar a " + eliminado);

    // Eliminar un elemento de la lista pasando como argumento el objeto.
    boolean resultado = lista.remove("Arya Stark");
    if (resultado) {
        System.out.println("Se ha eliminado a Arya Stark.");
    } else {
        System.out.println("No se ha encontrado a Arya Stark en la
        lista.");
    }

}
```

Y la lista quedaría así:

Índice	Elemento
0	"Eddard Stark"
1	"Jon Nieve"
2	"Sansa Stark"

## 6.5 Reemplazar un objeto de la lista

Método:

- E **set**(int indice, E e): Reemplaza el elemento que había en la posición por el que le pasamos como parámetro. Devuelve el elemento que se encontraba en dicha posición anteriormente.

Ejemplo:

```
// Sustituir el elemento en la posición 2 por Rob Stark.  
lista.set(2, "Robb Stark");
```

Tras ejecutar esta orden, la lista quedará así:

Índice	Elemento
0	"Eddard Stark"
1	"Jon Nieve"
2	"Rob Stark"

## 6.6 Comprobar si un objeto está dentro de la lista

Método:

- boolean **contains**(E e): Comprueba si el elemento especificado está en la colección.

Ejemplo:

```
// Comprobamos si Robb Stark está en la lista.  
boolean contiene = lista.contains("Robb Stark");  
if (contiene) {  
    System.out.println("Robb está en la lista.");  
} else {  
    System.out.println("Robb no está en la lista.");  
}
```

## 6.7 Buscar la posición de un elemento en el ArrayList

Métodos:

- `int indexOf(E e)`: Devuelve la primera posición en la que se encuentra el elemento *e*; -1 si no está.
- `int lastIndexOf(E e)`: Devuelve la última posición del elemento *e*; -1 si no está.

Ejemplos:

```
// Buscamos la primera aparición ocupada por un elemento.
int primeraPosicion = lista.indexOf("Jon Nieve");
System.out.println("Jon Nieve aparece en la posición " + primeraPosicion);

// Buscamos la última posición ocupada por un elemento.
int ultimaPosicion = lista.lastIndexOf("Jon Nieve");
System.out.println("Jon Nieve aparece por última vez en la posición " +
    ultimaPosicion);
```

Como Jon Nieve sólo está una vez en la lista, los dos prints anteriores mostrarán la misma posición.

## 6.8 Comprobar si una lista está vacía

Método:

- `boolean isEmpty()`: Devuelve *true* si la colección está vacía. Y *false* en caso contrario.

Ejemplo:

```
// Comprobamos si la lista está vacía.
if (lista.isEmpty())
    System.out.println("Lista vacía. La serie debe haber acabado.");
else
    System.out.println("Lista no vacía.");
```

## 6.9 Número de elementos de una lista

Método:

- `int size()`: Devuelve el número de elementos de la lista.

Ejemplo:

```
// Obtener cuántos elementos tiene la lista.  
System.out.println("Nuestra lista tiene " + lista.size() + " personajes.");
```

## 6.10 Vaciar una lista

Para borrar todos los elementos de una lista usaremos el método:

- `void clear()`: Elimina todos los elementos de la colección.

Ejemplo:

```
// Borramos la lista.  
lista.clear();
```

## 7. Recursividad

Un método recursivo es aquel que, dentro de su bloque de instrucciones, tiene alguna invocación a sí mismo. En él debe haber una selección múltiple entre dos casos:

- Caso base: En él se puede devolver directamente una respuesta.
- Caso recursivo: En él hay que ejecutar instrucciones que incluyen una llamada al propio método.

Puede verse una explicación bastante clara del concepto de recursividad en el siguiente [vídeo](#).

### 7.1 Algoritmo recursivo para calcular el factorial de un número

Cálculo del factorial de un número de manera iterativa y recursiva.

```
public static int factorial(int n) {  
    // No se puede calcular el factorial de un número negativo. Devolvemos -1  
    // para indicar que ha habido un error.  
    if (n < 0) {  
        return -1;  
    }  
  
    if (n > 0) {  
        // Caso recursivo.  
        return n * factorial(n - 1);  
    } else {  
        // Caso base cuando n vale 0.  
        return 1;  
    }  
}
```

## 7.2 Algoritmo recursivo para calcular la sucesión de Fibonacci

Versión recursiva del algoritmo para calcular la sucesión de Fibonacci (los índices empiezan desde cero):

```
public class Fibonacci {  
    public static void main(String[] args) {  
        for (int i = 0; i < 20; i++) {  
            System.out.println(fibonacci(i));  
        }  
    }  
  
    private static int fibonacci(int n) {  
        if (n > 1) {  
            // Caso recursivo.  
            return fibonacci(n - 1) + fibonacci(n - 2);  
        } else {  
            // Caso base cuando n vale 0 ó 1.  
            return 1;  
        }  
    }  
}
```