00.02 Java General

# 00.01 Ventanas de entrada de datos, mensajes y archivos

* [**Ventanas de diálogo con Swing**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#ventanas-de-di%C3%A1logo-con-swing)
  + [**1. Paquetes principales del API**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#1-paquetes-principales-del-api)
  + [**2. Componentes de Swing**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#2-componentes-de-swing)
  + [**3. JOptionPane**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#3-joptionpane)
    - [**showMessageDialog**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#showmessagedialog)
    - [**showInputDialog**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#showinputdialog)
  + [**4. JFileChooser**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#4-jfilechooser)
    - [**Directorio de trabajo**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#directorio-de-trabajo)
    - [**Selección de archivos y/o directorios**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#selecci%C3%B3n-de-archivos-yo-directorios)
    - [**Filtros de archivos**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/00dialogosenswing/#filtros-de-archivos)

## Ventanas de diálogo con Swing

**JFC (Java Foundation Classes)** es un conjunto de funciones/clases para **crear interfaces gráficas de usuario (GUI)** y añadir funcionalidad gráfica e interactividad a las aplicaciones Java que estudiaréis en la materia de ***Desenvolvemento de Interfaces***.

Incluye: **componentes gráficos de Swing** (botones, paneles, tablas, ventanas, etc.), **configuración de apariencia, API Java 2D, API de accesibilidad** (lectores de pantalla, Braille,…); **internacionalización** (gestión de idiomas del mundo,…)

En la materia de **Acceso a Datos** vamos a dar una **pequeña introducción a dos componentes de diálogo** que nos facilitarán la introducción de datos en la primera parte, mostrar mensajes o selección de ficheros hasta que lo estudiéis en otras materias:

* [***JOptionPane***](https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/JOptionPane.html)***:***
  + <https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/JOptionPane.html>
* [***JFileChooser***](https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/JFileChooser.html)
  + <https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/JFileChooser.html>

Una ventana de diálogo es una **subventana independiente** destinada a llevar un aviso temporal además de la ventana principal de la aplicación Swing.

Suelen usarse para **mostrar mensajes de error o una advertencia**, presentar imágenes, árboles de directorios, etc.

Para facilitar el trabajo, existen las clases de utilidad:

* [JOptionPane](https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/JOptionPane.html) para crear ventanas de **diálogos estándar y sencillos**.
* Barras de progreso: [JProgressBar](https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/JProgressBar.html) (ver también [ProgressMonitor](https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/ProgressMonitor.html))
* Elección de color: [JColorChooser](https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/JColorChooser.html).
* Selección de archivos: [JFileChooser](https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html).
* Diálogos de impresión: [API de impresión](https://docs.oracle.com/javase/tutorial/2d/printing/index.html).

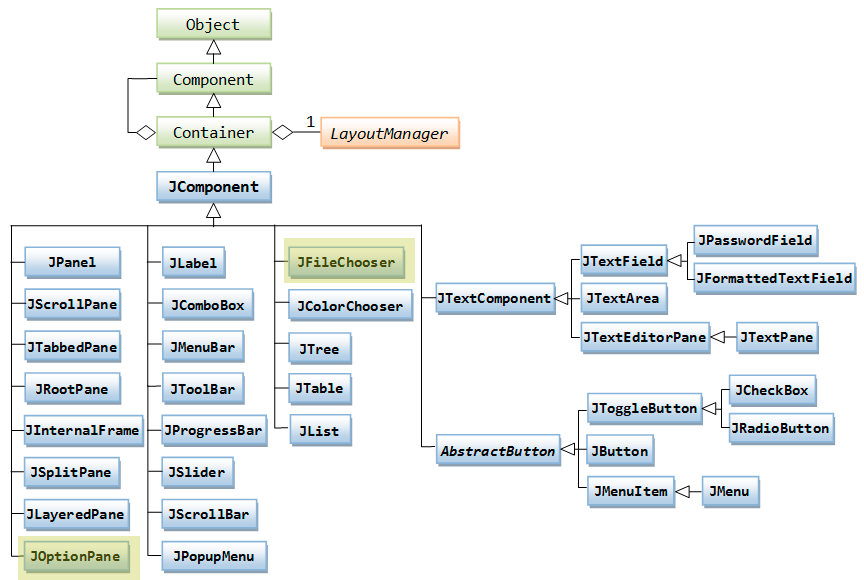
### 1. Paquetes principales del API

* javax.swing
* javax.swing.event

Además de:

|  |  |  |
| --- | --- | --- |
| javax.accessibility | javax.swing.plaf | javax.swing.text |
| javax.swing | javax.swing.plaf.basic | javax.swing.text.html |
| javax.swing.colorchooser | javax.swing.plaf.multi | javax.swing.text.rtf |
| javax.swing.border | javax.swing.plaf.metal | javax.swing.text.html.parser |
| javax.swing.event | javax.swing.plaf.synth | javax.swing.tree |
| javax.swing.filechooser | javax.swing.table | javax.swing.undo |

### 2. Componentes de Swing



### 3. JOptionPane

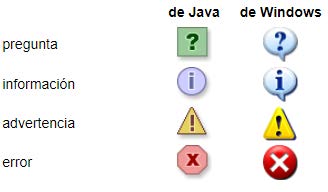
[**JOptionPane**](ttps://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/JOptionPane.html) permite crear y personalizar rápidamente varios tipos diferentes de **cuadros de diálogo**.

Proporciona soporte para diseñar **cuadros de diálogo estándar**, con iconos, especificar el título y el texto del cuadro de diálogo y personalizar el texto del botón.

La compatibilidad con iconos de le **permite especificar fácilmente qué icono muestra el cuadro de diálogo.**

**Son modales**, esto es, bloquea el acceso a la ventana padre.

Puede utilizar un **icono personalizado**, ningún icono o cualquiera de los **cuatro iconos estándar**:

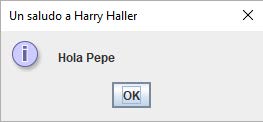
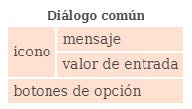


Cada apariencia tiene sus propias versiones de los cuatro íconos estándar.

La forma más sencilla de crear y mostrar diálogos con ***JOptionPane*** es por medio de los métodos ***showXxxDialog***:

|  |  |
| --- | --- |
| Método | Descripción |
| showConfirmDialog | Pregunta de confirmación, como sí/no/cancelar. |
| showInputDialog | Solicita entrada de datos. |
| showMessageDialog | Muestra un mensaje informativo |
| showOptionDialog | Permite personalizar el *JOptionPane*. |

*Existe una versión para marcos internos,****JInternalFrame****con métodos de la forma:****showInternalXxxx***

El formato del mensaje tiene una apariencia siguiendo la siguiente estructura:

Los parámetros de los métodos showXxxDialog son los siguientes:

* **parentComponent**: componente **padre**, el JFrame. Si el valor es **null**, se usa el JFrame por defecto y se centra en la pantalla.
* **mensaje**: **mensaje** de la ventana de diálogo. **Normalmente String**, pero puede ser cualquier objeto:
  + **Object []**: será interpretado como una serie de mensajes (uno por objeto) situados en vertical.
  + **Component**: mostrará el componente en la ventana de diálogo.
  + **Icon**: el icono será mostrado dentro de un JLabel.
  + **Otros**: se convierten a String llamando al método toString() y mostrándolo dentro de un JLabel.
* **Título**: título de la ventana.
* **messageType**: define el estilo del mensaje. Los posibles valores son:
  + ERROR\_MESSAGE
  + INFORMATION\_MESSAGE
  + WARNING\_MESSAGE
  + QUESTION\_MESSAGE
  + PLAIN\_MESSAGE
* **optionType**: conjunto **botones** de opción que aparen debajo de la ventana. Se pueden proporcionar otro botones usando el parámetro options.
  + DEFAULT\_OPTION
  + YES\_NO\_OPTION
  + YES\_NO\_CANCEL\_OPTION
  + OK\_CANCEL\_OPTION
* **options**: es una descripción más detallada del **conjunto de botones** que aparecen en la parte inferior de la ventana. Lo usual es un array de String, pero puede ser un array de Object:
  + **Component**: el componente se añade a la lista de botones.
  + **Icon:** se crea un JButton con este icono.
  + **Otros**: el objeto se convierte en String (toString()) y se emplea como etiqueta del botón.
* **Icono**: **icono** de la ventana de diálogo. El valor por defecto está determinado por el tipo de mensaje.
* **initialValue**: **valor por defecto** para ventanas de tipo input.

Los métodos **showXxxDialog** devuelven un **entero**, cuyos valores posibles son las constantes que referencian al botón pulsado:

* YES\_OPTION
* NO\_OPTION
* CANCEL\_OPTION
* OK\_OPTION
* CLOSED\_OPTION

Object[] opcionesBoton = {"Sí, por supuesto", "No, gracias", "No estoy loco!"};

int resultado = JOptionPane.showOptionDialog(this, "¿Te has vacunado de COVID?", // mensaje

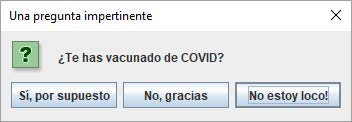
"Una pregunta impertinente", // título

JOptionPane.YES\_NO\_CANCEL\_OPTION, // OptionType

JOptionPane.QUESTION\_MESSAGE, // Tipo de mensaje null, // icono

opcionesBoton,

opcionesBoton[2]); // valor por defecto



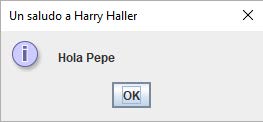
#### showMessageDialog

showMessageDialog : muestra un mensaje simple con un botón. (this es la referencia al formulario)

JOptionPane.showMessageDialog(this, "Hola Pepe",

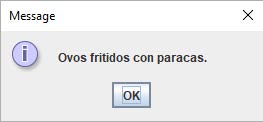
"Un saludo a Harry Haller",

JOptionPane.INFORMATION\_MESSAGE);



Con título e icono por defecto:

JOptionPane.showMessageDialog(this, "Ovos fritidos con paracas.");



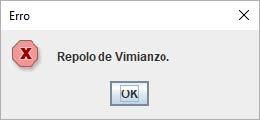
Con título, icono de aviso:

JOptionPane.showMessageDialog(this, "Arroz con chícharos.", "Aviso", JOptionPane.WARNING\_MESSAGE);



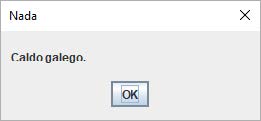
Con título, icono de error:

JOptionPane.showMessageDialog(this, "Repolo de Vimianzo.", "Erro", JOptionPane.ERROR\_MESSAGE);



Con título, sin icono:

JOptionPane.showMessageDialog(this, "Caldo galego.", "Nada", JOptionPane.PLAIN\_MESSAGE);



Con título, icono personalizado:

ImageIcon icoSalada = new ImageIcon( getClass().getResource("/images/ensalada.png"));

JOptionPane.showMessageDialog(this, "Salada con verde.", "Primeiro prato", JOptionPane.INFORMATION\_MESSAGE, iconaSalada);



#### showInputDialog

**showInputDialog** : es el único método **showXxxDialog** que no devuelve un entero.

**Devuelve un objeto**, normalmente un String:

Object[] platoFavorito = {"chícharos", "doce", "churros"};

String s = (String)JOptionPane.showInputDialog(this, "o meu prato favorito é: ",

"Introduce tu plato favorito", JOptionPane.PLAIN\_MESSAGE, iconaSalada, platoFavorito, "churros");



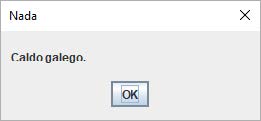
Si ponemos null en el array de opciones aparecerá una caja de texto:

**Devuelve un objeto**, normalmente un String:

String s = (String)JOptionPane.showInputDialog(this,

"o meu prato favorito é: ", "Introduce tu plato favorito", JOptionPane.PLAIN\_MESSAGE, iconaSalada,

null, "churros");



### 4. JFileChooser

Los **selectores de archivos proporcionan una GUI para navegar por el sistema de archivos y luego elegir un archivo o directorio de una lista**, o introducir el nombre de un archivo o directorio. } Normalmente usa la **clase JFileChooser para mostrar un cuadro de diálogo modal que contiene el selector de archivos**. Otra forma de presentar un selector de archivos es agregar una instancia de JFileChooser a un contenedor (ventana etc)

Ejemplo:

JFileChooser fc = new JFileChooser();

int returnVal = fc.showOpenDialog(this);

if (returnVal == JFileChooser.APPROVE\_OPTION) {

File file = fc.getSelectedFile();

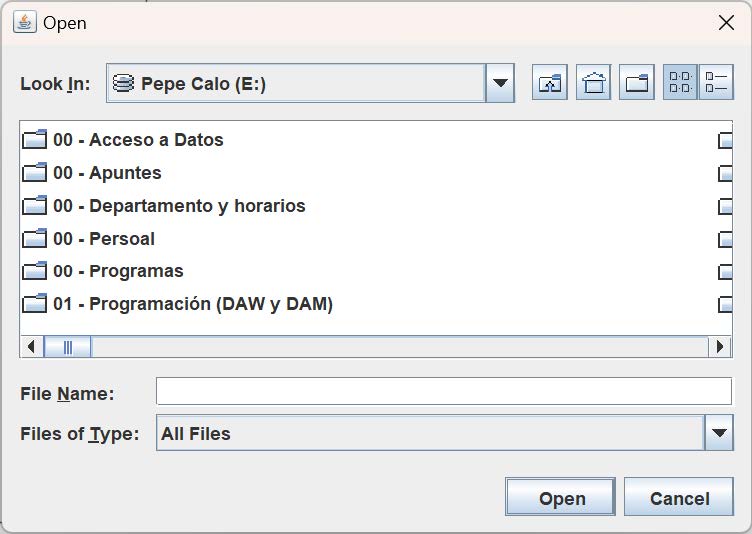
// Esto es lo que se hace con el archivo seleccionado

} else {

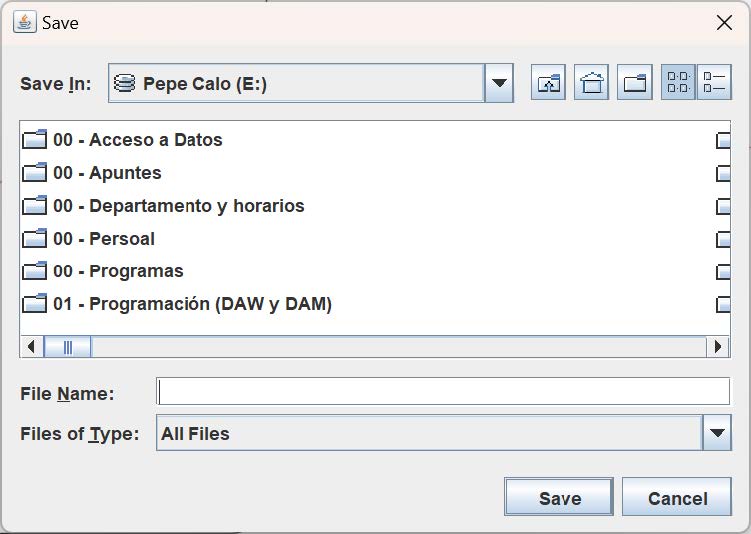
// Esto es lo que se hace si se cancela la selección

}

El método **showOpenDialog** muestra un cuadro de diálogo para abrir un archivo.



El método **showSaveDialog** muestra un cuadro de diálogo para guardar un archivo.



JFileChooser fcImaxe = new JFileChooser();

FileNameExtensionFilter filtro

= new FileNameExtensionFilter("Imágenes JPG y PNG", "jpg", "png");

fcImaxe.setFileFilter(filtro);

int valorSel = fcImaxe.showOpenDialog(null);

if (valorSel == JFileChooser.APPROVE\_OPTION){

System.out.println("Has seleccionado la imagen:" + fcImaxe.getSelectedFile().getName());

}

#### Directorio de trabajo

**showOpenDialog** recoge el **componente padre de la ventana de diálogo** y afecta a la posición de la ventana de diálogo.

Por **defecto muestra los archivos del directorio de trabajo del usuario**, pero puede especificarse el directorio inicial de varios modos:

* En el constructor:

JFileChooser fc = new JFileChooser("e:\\");

* Por medio del método:

fc.setCurrentDirectory(new File("e:\\"));

Selección de archivos y/o directorios

**showOpenDialog / showSaveDialog** recoge **devuelven un entero** que indica si se ha seleccionado un archivo: APPROVE\_OPTION o CANCEL\_OPTION

Una vez seleccionado un archivo o directorio (en ese caso debe indicarse que se permite selección de directorios) **puede invocarse al método getSelectedFile() para recuperar el archivo (File)**:

File arquivo = fcImaxe.getSelectedFile();

Una vez recuperado el archivo podemos obtener muchos datos del mismo (lo veremos en la unidad de archivos):

File arquivo = fcImaxe.getSelectedFile();

arquivo.getPath();

arquivo.getName();

arquivo.isDirectory();

arquivo.exists();

arquivo.delete();

// ...

Se puede utilizar **la misma instancia de la JFileChooser para mostrar un cuadro de diálogo estándar para guardar**.

int valor = fc.showSaveDialog(null);

Al utilizar la misma instancia del JFileChooser:

* **Recuerda el directorio actual entre usos**, por lo que las versiones para abrir y guardar comparten automáticamente el mismo directorio actual.
* **Sólo se personaliza un selector de archivos**, y las personalizaciones se aplican tanto a la versión para abrir como para guardar.

Se puede c**ambiar el modo de selección de archivos**, por ejemplo para seleccionar directorios:

fc.setFileSelectionMode(JFileChooser.DIRECTORIES\_ONLY);

Además, existen otros modos de selección:

* FILES\_AND\_DIRECTORIES.
* FILES\_ONLY (por defecto).
* DIRECTORIES\_ONLY.

#### Filtros de archivos

Por defecto, el **JFileChooser muestra todos los archivos y directorios, excepto los ocultos**.

Pueden programarse filtros de archivos para escoger algún tipo de archivo o directorio.

El JFileChooser **llama al método accept (de FileFilter) determina qué se mostrará**.

Existen varios tipos de filtros:

* **setFileHidingEnabled(false): para mostrar los archivos ocultos**.
* Subclases de la clase abstracta FileFilter: **FileNameExtensionFilter**: <https://docs.oracle.com/en/java/javase/22/docs/api/java.desktop/javax/swing/filechooser/FileNameExtensionFilter.html>

FileFilter filtro = new FileNameExtensionFilter("archivo JPEG", "jpg", "jpeg");

JFileChooser fc; // = ...;

fc.setFileFilter(filtro);

// fc.addChoosableFileFilter(filtro); // agrega a los seleccionables.

// fc. setAcceptAllFileFilterUsed(false);

# 00.02 Java Stream API

* [**Introducción**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#introducci%C3%B3n)
* [**1. Definición de Java Stream**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#1-definici%C3%B3n-de-java-stream)
* [**2. Procesamiento de Streams**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#2-procesamiento-de-streams)
* [**3. Crear un Stream**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#3-crear-un-stream)
* [**4. Operaciones Terminales y No Terminales**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#4-operaciones-terminales-y-no-terminales)
* [**5. Operaciones No Terminales**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#5-operaciones-no-terminales)
  + [**filter()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#filter)
  + [**map()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#map)
  + [**flatMap()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#flatmap)
  + [**distinct()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#distinct)
  + [**limit()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#limit)
  + [**peek()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#peek)
* [**6. Operaciones Terminales**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#6-operaciones-terminales)
  + [**anyMatch()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#anymatch)
  + [**allMatch()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#allmatch)
  + [**noneMatch()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#nonematch)
  + [**findFirst()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#findfirst)
  + [**findAny()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#findany)
  + [**forEach()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#foreach)
  + [**collect()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#collect)
  + [**count()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#count)
  + [**reduce()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#reduce)
  + [**min() y max()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#min-y-max)
  + [**forEachOrdered()**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#foreachordered)
* [**Resumen**](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/00repasojava/02javageneral/01streamapi/#resumen)

## Introducción

La API de Streams de Java ofrece un enfoque funcional para el procesamiento de colecciones de objetos. Se introdujo en Java 8 junto con varias otras características de programación funcional. Este tutorial de Java Stream explicará cómo funcionan estos streams funcionales y cómo usarlos.

La API de Java Stream no está relacionada con [Java InputStream](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/io/InputStream.html) y [Java OutputStream](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/io/OutputStream.html) de [Java IO](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/io/package-summary.html). InputStream y OutputStream se relacionan con flujos de bytes, mientras que la **API de Stream de Java se utiliza para procesar flujos de objetos**.

### 1. Definición de Java Stream

Un Stream en Java es un componente capaz de realizar una iteración interna de sus elementos, lo que significa que puede iterar sobre sus elementos por sí mismo. En contraste, al usar las características de iteración de [Java Collections](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/java-collections/index.html) (por ejemplo, un [Java Iterator](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/java-collections/iterator.html) o el bucle for-each de Java utilizado con un [Java Iterable](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/java-collections/iterable.html)), debes implementar la iteración de los elementos tú mismo.

### 2. Procesamiento de Streams

Puedes adjuntar oyentes a un Stream. Estos oyentes se llaman cuando el Stream itera internamente los elementos. Los oyentes se llaman una vez por cada elemento en el stream. De esta manera, cada oyente procesa cada elemento en el stream. Esto se denomina procesamiento de stream.

Los oyentes de un stream forman una cadena. El primer oyente en la cadena puede procesar el elemento en el stream y luego devolver un nuevo elemento para que el siguiente oyente en la cadena lo procese. Un oyente puede devolver el mismo elemento o uno nuevo, dependiendo del propósito de ese oyente (procesador).

### 3. Crear un Stream

Hay muchas formas de obtener un Stream en Java. Una de las formas más comunes de obtener un Stream es desde una [Java Collection](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Collection.html#stream()). Un ejemplo de cómo obtener un Stream desde una [Java List](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/List.html):

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

items.add("uno");

items.add("dos");

items.add("tres");

Stream<String> stream = items.stream();

Este ejemplo crea primero una lista de Java, luego agrega tres [Java Strings](https://manuais.pages.iessanclemente.net/plantillas/dam/ad/java/strings.html) y, finalmente, llama al método stream() para obtener una instancia de Stream.

### 4. Operaciones Terminales y No Terminales

La interfaz Stream tiene una selección de operaciones terminales y no terminales. Una operación no terminal de stream es una operación que agrega un oyente al stream sin hacer nada más. Una operación terminal de stream es una operación que inicia la iteración interna de los elementos, llama a todos los oyentes y devuelve un resultado.

Un ejemplo de Java Stream que contiene tanto una operación no terminal como una terminal:

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Stream;

public class EjemplosDeStream {

public static void main(String[] args) {

List<String> listaDeCadenas = new

ArrayList<>();

listaDeCadenas.add("Uno");

listaDeCadenas.add("Dos");

listaDeCadenas.add("Tres");

// Operación no terminal (filter)

Stream<String> streamFiltrado = listaDeCadenas.stream().filter(s -> s.startsWith("T"));

// Operación terminal (forEach)

streamFiltrado.forEach(System.out::println);

}

}

En este ejemplo, filter es una operación no terminal que filtra elementos basándose en el predicado proporcionado. Luego, forEach es una operación terminal que itera sobre los elementos restantes y aplica la función de impresión.

### 5. Operaciones No Terminales

Las operaciones no terminales de stream devuelven un nuevo Stream. Estas operaciones no realizan ninguna iteración interna. Se ejecutan “perezosamente”, lo que significa que no realizan ninguna acción hasta que se activa una operación terminal.

#### filter()

La operación filter es una **operación no terminal** que acepta un [Java Predicate](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/function/Predicate.html))como argumento y devuelve un nuevo Stream que contiene solo los elementos que cumplen con el predicado.

List<String> listaDeCadenas = Arrays.asList("Uno", "Dos", "Tres", "Cuatro", "Cinco");

// Filtrar elementos que comienzan con "C"

Stream<String> streamFiltrado = listaDeCadenas.stream().filter(s -> s.startsWith("C"));

// Imprimir elementos

streamFiltrado.forEach(System.out::println);

En este ejemplo, filter se utiliza para seleccionar solo las cadenas que comienzan con “C”.