

# DOCUMENTACIÓN

Librería de clases Java para la manipulación de datos tabulares.

Implementación modular orientada a objetos que permite cargar, filtrar, manipular, visualizar y gestionar datos provenientes de fuentes estructuradas.

## AUTORES

CÁCERES LUDMILA

FLORIDIA MICAELA

SILVETTI SANTINO

VILLANUEVA LEANDRO

ALGORITMOS I  
UNSAM  
LCD  
1C2025

<b>INTRODUCCIÓN</b>	<b>6</b>
<b>FUNCIONALIDADES PRINCIPALES</b>	<b>6</b>
<b>INFORMACIÓN BÁSICA</b>	<b>6</b>
<b>ACCESO INDEXADO</b>	<b>7</b>
<b>CARGA Y DESCARGA DE DATOS</b>	<b>7</b>
<b>VISUALIZACIÓN</b>	<b>7</b>
<b>GENERACIÓN Y MODIFICACIÓN</b>	<b>7</b>
<b>SELECCIÓN</b>	<b>8</b>
<b>FILTRADO</b>	<b>8</b>
<b>COPIA Y CONCATENACIÓN</b>	<b>8</b>
<b>ORDENAMIENTO</b>	<b>8</b>
<b>IMPUTACIÓN</b>	<b>8</b>
<b>MUESTREO</b>	<b>8</b>
<b>FUNCIONALIDADES OPCIONALES</b>	<b>9</b>
<b>AGREGACIÓN (GROUP BY)</b>	<b>9</b>
<b>PAQUETE: GESTIONDEDATOS</b>	<b>9</b>
<b>DESCRIPCIÓN</b>	<b>9</b>
<b>PRINCIPALES CLASES E INTERFACES</b>	<b>9</b>
TABLA	9
Métodos principales	10
Ejemplo de uso	10
COLUMNA	11
Métodos principales	11
Ejemplo de uso	11
CELDA	12
Métodos principales	12
Ejemplo de uso	12
Salida por consola	12
ETIQUETA (INTERFACE), ETIQUETASTRING, ETIQUETAENTERO	13
Métodos principales	13
Ejemplo de uso	13
Salida por consola	13
VALIDADORES DE TIPO DE DATO	14
Ejemplo de uso	14
Salida por consola	14
TIPODATO (ENUM)	15
<b>EJEMPLO DE USO MÁS COMPLETO</b>	<b>15</b>
Salida por consola	16
<b>RELACIÓN CON EL RESTO DEL SISTEMA</b>	<b>16</b>

<b>RESUMEN</b>	<b>17</b>
<b>FAQ</b>	<b>17</b>
<b>PAQUETE: ENTRADAYSALIDA</b>	<b>18</b>
<b>DESCRIPCIÓN</b>	<b>18</b>
<b>CLASE PRINCIPAL</b>	<b>18</b>
GESTORCSV	18
Métodos principales	18
Ejemplo de uso	19
Salida por consola	19
Relación con el resto del sistema	19
Resumen	20
FAQ	20
<b>PAQUETE: VISUALIZADOR</b>	<b>21</b>
<b>DESCRIPCIÓN</b>	<b>21</b>
<b>CLASE PRINCIPAL</b>	<b>21</b>
VISUALIZADOR	21
Métodos principales:	21
Ejemplo de uso	22
Salida por consola	22
<b>RELACIÓN CON EL RESTO DEL SISTEMA</b>	<b>23</b>
<b>RESUMEN</b>	<b>23</b>
<b>FAQ</b>	<b>23</b>
<b>PAQUETE: FILTROS</b>	<b>24</b>
<b>DESCRIPCIÓN GENERAL</b>	<b>24</b>
<b>ENUMERACIONES FUNDAMENTALES</b>	<b>24</b>
TIPOCOMPARADOR	24
OPERADORLOGICO	24
TIPOSINPARAMETRO	24
<b>INTERFACES PRINCIPALES</b>	<b>25</b>
FILTRO	25
FILTROLOGICO	25
FILTROSINPARAMETRO	25
<b>CLASES AUXILIARES Y FILTROS CONCRETOS (BÁSICOS)</b>	<b>25</b>
<b>EJEMPLO DE USO (FILTRADO SIMPLE)</b>	<b>26</b>
SALIDA POR CONSOLA	26
<b>FAQ (FILTROS SIMPLES)</b>	<b>26</b>
<b>FILTROS COMPUESTOS (LÓGICOS)</b>	<b>27</b>
FILTROAND	27
Ejemplo de uso	27

Salida por consola	27
FILTROOR	28
Ejemplo de uso	28
Salida por consola	28
FILTRONOT	29
Ejemplo de uso	29
Salida por consola	29
<b>CLASES Y FUNCIONES AUXILIARES AVANZADAS</b>	<b>29</b>
FILTROFACTORY	29
Métodos principales	29
TABLA PARCIAL	30
FILTRADO	30
Métodos principales	30
CONTROLADOR FILTRO	30
Métodos principales	30
EJEMPLO DE USO (FILTRADO COMPUESTO)	31
Salida por consola	31
<b>RELACIÓN CON EL RESTO DEL SISTEMA</b>	<b>32</b>
<b>RESUMEN DEL PAQUETE COMPLETO</b>	<b>32</b>
<b>FAQ (FILTROS AVANZADOS)</b>	<b>32</b>
 <b>PAQUETE: SELECCIÓN</b>	 <b>33</b>
 <b>DESCRIPCIÓN GENERAL</b>	 <b>33</b>
<b>CLASE PRINCIPAL</b>	<b>33</b>
SELECCION	33
CONSTRUCTOR	33
<b>MÉTODOS PRINCIPALES</b>	<b>33</b>
HEAD (INT NUMFILAS)	33
TAIL (INT NUMFILAS)	33
SELECCIONAR(LIST<ETIQUETA> ETIQUETA FILAS, LIST<ETIQUETA> ETIQUETAS COLUMNAS)	34
<b>EJEMPLO DE USO</b>	<b>34</b>
SALIDA POR CONSOLA	35
<b>RELACIÓN CON EL RESTO DEL SISTEMA</b>	<b>35</b>
<b>RESUMEN</b>	<b>35</b>
<b>FAQ</b>	<b>36</b>
 <b>PAQUETE: COPIAYCONCATENACION</b>	 <b>37</b>
 <b>DESCRIPCIÓN GENERAL</b>	 <b>37</b>
<b>CLASES PRINCIPALES</b>	<b>37</b>
COPIA TABLA	37
Método principal	37
CONCATENACION TABLA	37
Método principal	37

<b>EJEMPLO DE USO</b>	<b>38</b>
SALIDA POR CONSOLA	38
<b>RELACIÓN CON EL RESTO DEL SISTEMA</b>	<b>39</b>
<b>RESUMEN</b>	<b>39</b>
<b>FAQ</b>	<b>39</b>
 <b>PAQUETE: MANIPULACION</b>	 <b>40</b>
 <b>DESCRIPCIÓN GENERAL</b>	 <b>40</b>
<b>CLASES PRINCIPALES</b>	<b>40</b>
ORDENADORDATOS	40
Constructor	40
Método principal	40
EJEMPLO DE USO	41
Salida por consola	42
<b>MUESTREADORDATOS</b>	<b>42</b>
CONSTRUCTOR	42
MÉTODO PRINCIPAL	42
EJEMPLO DE USO	43
Salida por consola	44
<b>IMPUTADORDATOS</b>	<b>44</b>
CONSTRUCTOR	44
MÉTODO PRINCIPAL	44
EJEMPLO DE USO	45
Salida por consola	46
<b>RELACIÓN CON EL RESTO DEL SISTEMA</b>	<b>46</b>
<b>RESUMEN</b>	<b>46</b>
<b>FAQ</b>	<b>47</b>
 <b>PAQUETE: GESTION DE ERRORES</b>	 <b>48</b>
 <b>DESCRIPCIÓN GENERAL</b>	 <b>48</b>
<b>CLASES PRINCIPALES</b>	<b>48</b>
GESTIONERRORES	48
Métodos principales	48
EXCEPCIONTABULAR (ABSTRACTA)	48
EXCEPCIONVALIDACION	48
EXCEPCIONOPERACIONNoVALIDA	48
EXCEPCIONCARGADATOS	49
<b>NOTA</b>	<b>49</b>
<b>EJEMPLO DE USO</b>	<b>49</b>
<b>RELACIÓN CON EL RESTO DEL SISTEMA</b>	<b>49</b>
<b>RESUMEN</b>	<b>49</b>
<b>FAQ</b>	<b>49</b>

<b>PAQUETE: HERRAMIENTAS</b>	<b>50</b>
<b>DESCRIPCIÓN GENERAL</b>	<b>50</b>
<b>CLASE PRINCIPAL</b>	<b>50</b>
MEDIDOREFICIENCIA	50
MÉTODOS PRINCIPALES	50
medirTiempo(string descripcion, runnable tarea)	50
marcarInicio()	50
mostrarTiempoTotal(long inicio):	50
<b>RELACIÓN CON EL RESTO DEL SISTEMA</b>	<b>51</b>
<b>RESUMEN</b>	<b>51</b>
<b>FAQ</b>	<b>51</b>
<b>PAQUETE: AGREGACION</b>	<b>52</b>
<b>DESCRIPCIÓN GENERAL</b>	<b>52</b>
<b>PRINCIPALES CLASES Y COMPONENTES</b>	<b>52</b>
TIPOOPERACION (ENUM)	52
AGREGADORTABLA	52
TABLAAGRUPADA	52
SUMARIZADOR (INTERFACE)	52
Implementaciones de sumador	53
FABRICAOPERACIONESSUMARIZACION	53
MÉTODOS PRINCIPALES	53
<b>EJEMPLO DE USO</b>	<b>54</b>
SALIDA POR CONSOLA	55
<b>RELACIÓN CON EL RESTO DEL SISTEMA</b>	<b>55</b>
<b>RESUMEN</b>	<b>55</b>
<b>FAQ</b>	<b>56</b>
<b>NOTAS FINALES</b>	<b>57</b>
<b>CRÉDITOS</b>	<b>57</b>
<b>AGRADECIMIENTOS</b>	<b>57</b>

## INTRODUCCIÓN

La presente librería , fue diseñada como proyecto final integrador para la materia ALGORITMOS I (Universidad Nacional de San Martín) y reúne los conceptos fundamentales abordados durante la cursada, tomando como punto de partida un problema real que se resuelve mediante un análisis inicial del problema que queremos abordar, el diseño de una solución y su posterior implementación en Java con orientación a objetos. El desarrollo del trabajo se organizó en etapas de entregas parciales para facilitar el seguimiento.

## FUNCIONALIDADES PRINCIPALES

### INFORMACIÓN BÁSICA

La librería permite obtener, para una estructura tabular dada:

- Cantidad de filas
- Cantidad de columnas
- Etiquetas de filas
- Etiquetas de columnas
- Tipos de datos de cada columna

Cada columna tiene un tipo de dato asociado, que se valida al momento de su creación o modificación. Los tipos de datos soportados son:

- Numérico (entero, real, etc.)
- Booleano
- Cadena

Existe un valor especial para representar valores faltantes (NA), que puede asignarse a cualquier celda, sin importar el tipo de dato.

La librería está diseñada para Java 8 o superior.

## ACCESO INDEXADO

Se provee acceso indexado a nivel de fila y columna, permitiendo:

- Acceder a una fila completa mediante su etiqueta
- Acceder a una columna completa mediante su etiqueta
- Acceder a una celda mediante las etiquetas de fila y columna

Las etiquetas pueden ser numéricas o cadenas. Si no se especifican, se inicializan como una secuencia de enteros desde 0.

## CARGA Y DESCARGA DE DATOS

La librería soporta la lectura y escritura de datos en formato CSV, tanto en memoria como en disco. Se puede establecer el carácter delimitador de columnas y la utilización de encabezados.

## VISUALIZACIÓN

Se ofrece una visualización sencilla en formato texto, configurable para limitar la cantidad de filas, columnas o caracteres por celda que se muestran en caso de estructuras grandes.

## GENERACIÓN Y MODIFICACIÓN

La estructura tabular puede generarse a partir de:

- Un archivo CSV
- Copia profunda de otra estructura tabular
- Una estructura bidimensional nativa de Java
- Una secuencia lineal nativa de Java

Se permite modificar la estructura:

- Asignando valores a celdas individuales
- Insertando o eliminando columnas y filas
- Insertando columnas desde otra columna o desde una secuencia lineal de Java



## SELECCIÓN

Permite la selección parcial de filas y/o columnas a través de listas de etiquetas, generando una **vista** reducida (no una copia). Incluye operaciones como:

- **head(x)**: primeras x filas
- **tail(x)**: últimas x filas

## FILTRADO

Permite la selección parcial de filas mediante filtros aplicados sobre los valores de las celdas, combinando comparadores y operadores lógicos.

## COPIA Y CONCATENACIÓN

- **Copia independiente**: Crea una copia profunda de la estructura.
- **Concatenación**: Genera una nueva estructura tabular concatenando dos existentes (si coinciden las columnas en cantidad, orden, tipo de dato y etiqueta).

## ORDENAMIENTO

Permite ordenar filas por uno o más criterios (columnas), en orden ascendente o descendente.

## IMPUTACIÓN

Permite rellenar valores faltantes (NA) con un valor literal indicado.

## MUESTREO

Selecciona aleatoriamente un porcentaje de las filas.

## FUNCIONALIDADES OPCIONALES

### AGREGACIÓN (GROUP BY)

Divide filas en grupos según el valor de una o más columnas, se pueden aplicar operaciones estadísticas (suma, máximo, mínimo, cuenta, media, varianza, desvío estándar) y devolver una estructura donde las filas representen los grupos y las columnas los resultados de las operaciones.

## PAQUETE: GESTIONDEDATOS

### DESCRIPCIÓN

El paquete `gestiondedatos` contiene las clases e interfaces centrales para la representación y validación de los datos tabulares en la librería. Aquí se definen los conceptos fundamentales de tabla, columna, celda y etiquetas, así como los validadores de tipo de dato y las enumeraciones asociadas. Este paquete constituye el núcleo sobre el cual operan el resto de los módulos y garantiza la consistencia y flexibilidad de las operaciones.

### PRINCIPALES CLASES E INTERFACES

---

#### TABLA

Esta es la clase principal que representa una estructura de datos tabular, donde los datos están organizados en filas y columnas, permite crear tablas, agregar/eliminar filas o columnas, acceder a celdas, modificar datos y operar con funcionalidades avanzadas integrando módulos externos (selección, filtrado, visualización, etc.).

## MÉTODOS PRINCIPALES

- Constructor por defecto / con etiquetas y tipos de columna
  - Permite crear tablas vacías o con estructura personalizada
- agregarColumna(), eliminarColumna(): Añadir/eliminar columnas.
- agregarFila(), eliminarFila(): Añadir/eliminar filas.
- getCelda(), getColumna(), getFila: Acceso a datos individuales o por conjunto.
- visualizarTabla(), infoTabla(): Visualización en consola.
- Métodos de integración con otros módulos: copia, concatenación, imputación, muestreo, ordenamiento, selección, filtrado.

## EJEMPLO DE USO

```
//Creación de una tabla con 2 columnas: "Nombre" (cadena) y "Edad" (numérico)
```

```
List<Etiqueta> etiquetas = Arrays.asList(
    new EtiquetaString("Nombre"),
    new EtiquetaString("Edad")
);
List<TipoDato> tipos = Arrays.asList(
    TipoDato.CADENA,
    TipoDato.NUMERICO
);
Tabla tabla = new Tabla(etiquetas, tipos);
```

```
//Agregado de una fila
```

```
List<Celda<?>> fila = Arrays.asList(
    new Celda<String>("Ana", TipoDato.CADENA),
    new Celda<Integer>(22, TipoDato.NUMERICO)
);
tabla.agregarFila(fila);
```

```
//Visualización
```

```
tabla.visualizarTabla();
```

## SALIDA POR CONSOLA

FILA	Nombre	Edad
0	Ana	22

---

## COLUMNA

Representa una columna de la tabla, con un tipo de dato homogéneo. Permite agregar/eliminar celdas y consultar información sobre la columna.

---

### MÉTODOS PRINCIPALES

- `Columna(TipoDato tipoDato)`, `Columna(TipoDato tipo, List<Celda<?>>)`: Constructores.
- `agregarCelda()`, `eliminarCelda()`: Gestión de celdas.
- `getTipoDato()`, `getCeldas()`, `contarCeldas()`: Consultas.
- `copiar()`: Copia profunda de la columna.

---

### EJEMPLO DE USO

```
Columna col = new Columna(TipoDato.NUMERICO);

//Agregamos una celda numérica

col.agregarCelda(new Celda<Integer>(20, TipoDato.NUMERICO));
col.agregarCelda(new Celda<Integer>(35, TipoDato.NUMERICO));

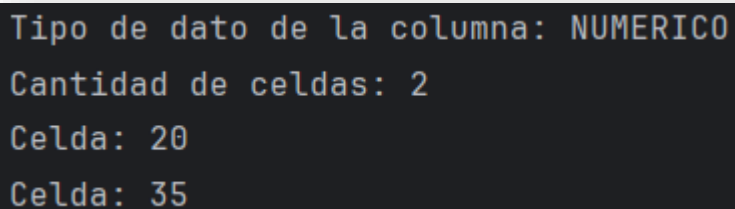
//Imprimimos el contenido de la columna

System.out.println("Tipo de dato de la columna: " + col.getTipoDato());
System.out.println("Cantidad de celdas: " + col.contarCeldas());

for (Celda<?> celda : col.getCeldas()) {
    System.out.println("Celda: " + celda);
}
```

---

### SALIDA POR CONSOLA



```
Tipo de dato de la columna: NUMERICO
Cantidad de celdas: 2
Celda: 20
Celda: 35
```

---

## CELDA

Contiene un valor concreto (numérico, booleano, cadena o NA) y su tipo. Permite saber si es NA y copiarse.

---

### MÉTODOS PRINCIPALES

- `Celda(T valor, TipoDato tipo)`: Constructor.
- `getValor()`, `getTipoDato()`, `isNA()`: Consultas.
- `copiar()`: Copia profunda.

---

### EJEMPLO DE USO

```
//Creamos una celda numérica

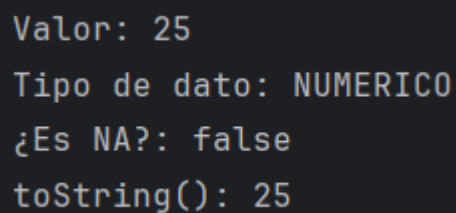
Celda<Integer> celdaEdad = new Celda<>(25, TipoDato.NUMERICO);

//Imprimimos información de la celda

System.out.println("Valor: " + celdaEdad.getValor());
System.out.println("Tipo de dato: " + celdaEdad.getTipoDato());
System.out.println("¿Es NA?: " + celdaEdad.isNA());
System.out.println("toString(): " + celdaEdad);
```

---

### SALIDA POR CONSOLA



```
Valor: 25
Tipo de dato: NUMERICO
¿Es NA?: false
toString(): 25
```

## ETIQUETA (INTERFACE), ETIQUETASTRING, ETIQUETAENTERO

Representan etiquetas de filas y columnas, permitiendo que puedan ser enteros (posición) o cadenas (nombres).  
Garantizan identificación y flexibilidad de acceso.

### MÉTODOS PRINCIPALES

- `getValor()`: Obtiene el valor asociado.
- `copiar()`: Copia independiente.

### EJEMPLO DE USO

```
//Creamos las etiquetas

Etiqueta<String> colNombre = new EtiquetaString("Nombre");
Etiqueta<Integer> filaUno = new EtiquetaEntero(1);

//Imprimimos información de las etiquetas

System.out.println("Etiqueta de columna: " + colNombre.getValor());
System.out.println("Etiqueta de fila: " + filaUno.getValor());
System.out.println("Etiqueta columna toString: " + colNombre);
System.out.println("Etiqueta fila toString: " + filaUno);

//Copiamos las etiquetas

Etiqueta<String> copiaColNombre = colNombre.copiar();
Etiqueta<Integer> copiaFilaUno = filaUno.copiar();

System.out.println("¿Copia de columna es igual al original?: " + copiaColNombre.equals(colNombre));
System.out.println("¿Copia de fila es igual al original?: " + copiaFilaUno.equals(filaUno));
```

### SALIDA POR CONSOLA

```
Etiqueta de columna: Nombre
Etiqueta de fila: 1
Etiqueta columna toString: Nombre
Etiqueta fila toString: 1
¿Copia de columna es igual al original?: true
¿Copia de fila es igual al original?: true
```

## VALIDADORES DE TIPO DE DATO

- ValidadorTipoDato: Clase utilitaria que asocia cada tipo de dato (TipoDato) con un validador específico y valida datos al agregar/modificar celdas.
- TipoValidador (interface): Interfaz para validadores.
- ValidadorString, ValidadorNumerico, ValidadorBooleano, ValidadorNA: Implementaciones para cada tipo de dato soportado.

## EJEMPLO DE USO

```
//Ejemplo de validaciones

boolean valido1 = ValidadorTipoDato.esValido("texto", TipoDato.CADENA, TipoDato.CADENA);
boolean valido2 = ValidadorTipoDato.esValido(123, TipoDato.NUMERICO, TipoDato.NUMERICO);
boolean valido3 = ValidadorTipoDato.esValido(true, TipoDato.BOOLEANO, TipoDato.BOOLEANO);
boolean valido4 = ValidadorTipoDato.esValido(null, TipoDato.CADENA, TipoDato.NA);
boolean valido5 = ValidadorTipoDato.esValido("no es numero", TipoDato.NUMERICO, TipoDato.CADENA);

System.out.println("¿'texto' es válido para CADENA?: " + valido1);    // true
System.out.println("¿123 es válido para NUMERICO?: " + valido2);    // true
System.out.println("¿true es válido para BOOLEANO?: " + valido3);   // true
System.out.println("¿null es válido para NA?: " + valido4);         // true
System.out.println("¿'no es numero' es válido para NUMERICO?: " + valido5); // false
```

## SALIDA POR CONSOLA

```
¿'texto' es válido para CADENA?: true
¿123 es válido para NUMERICO?: true
¿true es válido para BOOLEANO?: true
¿null es válido para NA?: true
¿'no es numero' es válido para NUMERICO?: false

Process finished with exit code 0
```

## TIPODATO (ENUM)

Enumeración que define los tipos de datos soportados por columnas y celdas: NUMERICO, CADENA, BOOLEANO, NA.

### EJEMPLO DE USO MÁS COMPLETO

El siguiente ejemplo ilustra la creación y estructura flexible de tablas, la inserción y modificación de datos, el acceso por etiqueta y posición, la copia profunda e independencia de datos y la consistencia y manejo de filas/columnas:

```
//1ro Definimos las etiquetas de columnas y sus tipos
List<Etiqueta> etiquetas = Arrays.asList( new EtiquetaString("Nombre"), new EtiquetaString("Edad"), new EtiquetaString("Graduado"));
List<TipoDato> tipos = Arrays.asList( TipoDato.CADENA, TipoDato.NUMERICO, TipoDato.BOOLEANO );

//2do Creamos la tabla con esa estructura
Tabla tabla = new Tabla(etiquetas, tipos);

//3ro Agregamos filas de datos
List<Celda<?>> fila1 = Arrays.asList(
    new Celda<String>("Ana", TipoDato.CADENA), new Celda<Integer>(22, TipoDato.NUMERICO), new Celda<Boolean>(true, TipoDato.BOOLEANO) );
List<Celda<?>> fila2 = Arrays.asList(
    new Celda<String>("Juan", TipoDato.CADENA), new Celda<Integer>(27, TipoDato.NUMERICO), new Celda<Boolean>(false, TipoDato.BOOLEANO));

tabla.agregarFila(fila1);
tabla.agregarFila(fila2);

//4to Visualización de la tabla
System.out.println("Tabla original:");
tabla.visualizarTabla();

//5to acceso a una celda específica
Etiqueta colEdad = new EtiquetaString("Edad");
Etiqueta fila0 = new EtiquetaEntero(0); // Fila "Ana" porque es la primera
Celda<?> celdaEdadAna = tabla.getCelda(colEdad, fila0);
System.out.println("\nEdad de Ana: " + celdaEdadAna.getValor());

//6to modificación de un dato (cambiar la edad de Juan)
Etiqueta fila1Etiqueta = new EtiquetaEntero(1); // Fila "Juan"
Celda<Integer> nuevaEdadJuan = new Celda<>(28, TipoDato.NUMERICO);
tabla.getColumna(colEdad).getCeldas().set(1, nuevaEdadJuan);

//7mo visualizamos de nuevo para ver el cambio
System.out.println("\nTabla después de modificar la edad de Juan:");
tabla.visualizarTabla();

//8vo creamos una copia profunda e independiente
Tabla copia = tabla.copiarTabla();
System.out.println("\nCopia independiente de la tabla:");
copia.visualizarTabla();

//9no Eliminamos la fila de Ana en la tabla original
tabla.eliminarFila(fila0);

//10mo mostramos ambas tablas para ver que son independientes
System.out.println("\nTabla original tras eliminar la fila de Ana:");
tabla.visualizarTabla();
System.out.println("\nCopia (sin modificaciones):");
copia.visualizarTabla();
```



## SALIDA POR CONSOLA

Tabla original:

FILA	Nombre	Edad	Graduado
0	Ana	22	true
1	Juan	27	false

Edad de Ana: 22

Tabla después de modificar la edad de Juan:

FILA	Nombre	Edad	Graduado
0	Ana	22	true
1	Juan	28	false

Copia independiente de la tabla:

FILA	Nombre	Edad	Graduado
0	Ana	22	true
1	Juan	28	false

Tabla original tras eliminar la fila de Ana:

FILA	Nombre	Edad	Graduado
1	Juan	28	false

Copia (sin modificaciones):

FILA	Nombre	Edad	Graduado
0	Ana	22	true
1	Juan	28	false

## RELACIÓN CON EL RESTO DEL SISTEMA

El paquete `gestiondedatos` es el núcleo central de la librería y se integra con todos los demás módulos. Las clases acá definidas:

- Son utilizadas directamente por los paquetes de manipulación, selección, visualización, copiado, concatenación y filtrado.
- Las operaciones avanzadas (imputación, muestreo, ordenamiento, etc.) dependen de la estructura y validación que proveen estas clases.
- El modelo de datos consistente garantiza que otras funcionalidades puedan operar sin riesgo de errores de tipo o inconsistencias.

## RESUMEN

- `gestiondedatos` define la estructura fundamental de la tabla y sus componentes básicos.
- Permite flexibilidad y seguridad en la manipulación de datos tabulares.
- El diseño orientado a objetos y el uso de validadores garantizan la integridad de los datos durante todas las operaciones.

## FAQ

### ¿PUEDO USAR ETIQUETAS NUMÉRICAS Y DE TEXTO A LA VEZ?

Sí, podés tener columnas o filas identificadas por números o cadenas de texto según lo que más te convenga.

### ¿QUÉ PASA SI INTENTO PONER UN DATO DE TIPO INCORRECTO EN UNA COLUMNA?

El sistema lanzará una excepción de validación y evitará que se agregue ese valor incompatible.

### ¿SE PUEDE TENER UNA CELDA SIN VALOR?

Sí, usando el tipo de dato `NA` para marcar valores faltantes.

### ¿LA TABLA COPIA LOS DATOS O LOS REFERENCIA?

Las operaciones de copia profunda (`copiarTabla`, `copiar en columna/celda`) aseguran que se genere una estructura totalmente independiente.

### ¿POR QUÉ SE USAN VALIDADORES POR TIPO?

Para garantizar que no se inserten datos incompatibles y mantener la coherencia de los tipos de datos en toda la tabla.

### ¿CÓMO SE GUARDA O CARGA UNA TABLA DESDE UN ARCHIVO?

Utilizá los métodos del paquete de entrada/salida de datos para leer o escribir la tabla en formato CSV. El núcleo `gestiondedatos` sólo define la estructura; la persistencia está delegada a otro módulo.

### ¿QUÉ SIGNIFICA NA Y CUÁNDO DEBERÍA USARLO?

`NA` representa un valor faltante. Debe usarse cuando no se dispone de información para una celda, independientemente del tipo de dato de la columna.

### ¿QUÉ OCURRE SI INTENTO AGREGAR UNA COLUMNA O FILA CON UNA ETIQUETA QUE YA EXISTE?

El sistema arrojará una excepción para evitar duplicados y mantener la integridad de la tabla.

## PAQUETE: ENTRADAYSALIDA

### DESCRIPCIÓN

El paquete entradaysalida proporciona las clases necesarias para la importación y exportación de datos tabulares en formato CSV. Permite transformar archivos externos en instancias de la clase Tabla y viceversa, gestionando automáticamente la inferencia de tipos de datos, valores faltantes (NA) y configuraciones comunes de archivos CSV como el delimitador de columnas y la presencia de encabezados. Es el punto de entrada y salida de la librería.

### CLASE PRINCIPAL

#### GESTORCSV

Esta clase es el gestor principal de archivos CSV de la librería. Se encarga de cargar y guardar estructuras tabulares en archivos, infiriendo tipos de datos, manejando valores faltantes y facilitando la interoperabilidad entre el mundo externo (archivos) y las estructuras internas de la librería.

#### MÉTODOS PRINCIPALES

- Constructor por defecto  
Inicializa el gestor con delimitador , y encabezado activado por defecto.
- setDelimitador(String delimitador)  
Permite cambiar el carácter delimitador de columnas (por ejemplo, usar punto y coma ;).
- setUsarEncabezado(boolean usarEncabezado)  
Indica si la primera fila es tomada como encabezado (nombres de columnas).
- Tabla cargarCSV(String rutaArchivo)  
Carga un archivo CSV, infiere tipos y genera una instancia de Tabla.  
Lanza excepciones si ocurre un error de lectura o inconsistencia en los datos.
- void guardarCSV(String rutaArchivo, Tabla tabla)  
Guarda una tabla en un archivo CSV, usando la configuración actual de delimitador y encabezado.

---

## EJEMPLO DE USO

test\_datos.csv

```
Nombre,Edad,Activo
Ana,28,true
Juan,35,false
Luis,22,true
```

```
GestorCSV gestor = new GestorCSV();
gestor.setDelimitador(","); //delimitador por defecto
gestor.setUsarEncabezado(true); //Leer/guardar encabezados
```

```
//cargamos una tabla desde archivo CSV
```

```
Tabla tabla = gestor.cargarCSV("test_datos.csv");
```

```
//visualizamos la tabla cargada
```

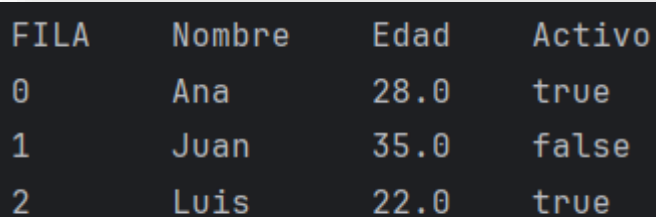
```
tabla.visualizarTabla();
```

```
//guardamos la tabla (o una copia) en un nuevo archivo
```

```
gestor.guardarCSV("salida_ordenada.csv", tabla);
```

---

## SALIDA POR CONSOLA



FILA	Nombre	Edad	Activo
0	Ana	28.0	true
1	Juan	35.0	false
2	Luis	22.0	true

---

## RELACIÓN CON EL RESTO DEL SISTEMA

- GestorCSV es el puente entre archivos CSV y la clase Tabla (núcleo de la librería).
- Colabora con el paquete gestiondedatos para construir la estructura tabular y gestionar validaciones.
- Depende del paquete gestiondeerrores para notificar problemas en la carga o escritura.
- Es usualmente el primer paso para trabajar con datos reales, previo a aplicar filtros, selecciones, visualizaciones, etc.

---

## RESUMEN

- Permite la carga y guardado de datos tabulares desde/hacia archivos CSV.
- Realiza inferencia automática de tipos de dato por columna.
- Permite gestionar valores faltantes, delimitadores personalizados y encabezados.
- Facilita la interoperabilidad entre la librería y archivos externos.

---

## FAQ

### ¿QUÉ OCURRE SI UNA CELDA ESTÁ VACÍA EN EL CSV?

*Se interpreta automáticamente como un valor faltante (NA) dentro de la tabla.*

### ¿PUEDO USAR DELIMITADORES DISTINTOS A LA COMA?

*Sí, podés configurarlo con `setDelimitador(String delimitador)`.*

### ¿CÓMO DEFINE EL TIPO DE CADA COLUMNA AL CARGAR EL ARCHIVO?

*Se infiere automáticamente: si todos los valores son numéricos, la columna es NUMÉRICA; si todos son “true” o “false”, es BOOLEANO; si hay valores mixtos, se considera CADENA.*

### ¿QUÉ PASA SI HAY UN VALOR INCONSISTENTE EN UNA COLUMNA?

*El gestor asigna NA a la celda y emite una advertencia, pero sigue procesando el resto del archivo.*

### ¿CÓMO GUARDO UNA TABLA EN UN NUEVO ARCHIVO CSV?

*Usá `guardarCSV(ruta, tabla)`. Se exportan los datos actuales de la tabla al archivo.*

## PAQUETE: VISUALIZADOR

### DESCRIPCIÓN

El paquete visualizador ofrece utilidades para la presentación visual y legible de los datos tabulares en la consola. Permite imprimir una tabla en un formato más amigable, ajustando el ancho de columnas, truncando el contenido si es necesario y limitando la cantidad de filas y columnas para garantizar una experiencia más cómoda en terminales o consolas que pudieran tener el ancho restringido. Incluye también métodos para mostrar información resumen de la estructura y los tipos de dato de una tabla.

### CLASE PRINCIPAL

---

#### VISUALIZADOR

Clase utilitaria que permite mostrar en pantalla una tabla o información relevante de la misma de forma ordenada y configurable.

---

#### MÉTODOS PRINCIPALES:

- `static void imprimirTabla(Tabla tabla)`
  - Imprime la tabla en formato de texto, con ajuste automático de columnas, truncado de celdas y límite de filas visibles. Indica también si hay filas o columnas ocultas por limitaciones en el ancho.
- `static void info(Tabla tabla)`
  - Muestra información básica sobre la tabla: como cantidad de filas, cantidad de columnas y tipos de dato de cada columna.

## EJEMPLO DE USO

```
//Creamos una tabla sencilla
List<Etiqueta> etiquetas = Arrays.asList(
    new EtiquetaString("Producto"),
    new EtiquetaString("Precio"),
    new EtiquetaString("Stock")
);
List<TipoDato> tipos = Arrays.asList(
    TipoDato.CADENA,
    TipoDato.NUMERICO,
    TipoDato.NUMERICO
);
Tabla tabla = new Tabla(etiquetas, tipos);

tabla.agregarFila(Arrays.asList(
    new Celda<>("Café en grano 1kg", TipoDato.CADENA),
    new Celda<>(4100, TipoDato.NUMERICO),
    new Celda<>(20, TipoDato.NUMERICO)
));

tabla.agregarFila(Arrays.asList(
    new Celda<>("Yerba 500g saborizada", TipoDato.CADENA),
    new Celda<>(2000, TipoDato.NUMERICO),
    new Celda<>(15, TipoDato.NUMERICO)
));

//hacemos la visualización principal
Visualizador.imprimirTabla(tabla);

//mostramos información básica
Visualizador.info(tabla);
```

## SALIDA POR CONSOLA

```
FILA  Producto      Precio  Stock
0     Café en gran...  4100   20
1     Yerba 500g s...  2000   15

----- Información de la Tabla -----
Cantidad de Filas: 2
Cantidad de Columnas: 3

- Producto: CADENA
- Precio: NUMERICO
- Stock: NUMERICO
```

## RELACIÓN CON EL RESTO DEL SISTEMA

- Es el principal canal de salida visual para cualquier tabla generada o modificada con la librería.
- Se integra directamente con el núcleo gestiondedatos.Tabla, accediendo a filas, columnas, celdas y sus etiquetas.
- Suele ser usado en conjunto con los módulos de entrada/salida, manipulación y filtrado para verificar resultados parciales o finales.

## RESUMEN

- Permite visualizar tablas de forma clara, sin depender de interfaces gráficas.
- Ajusta el formato y limita el contenido para adaptarse a consolas tradicionales.
- Ofrece información estructural útil para diagnóstico rápido de los datos cargados o procesados.

## FAQ

### ¿PUEDO MOSTRAR TABLAS MUY GRANDES?

*Sí, pero solo se muestran hasta 20 filas y las columnas que entren en el ancho máximo configurado. Se indica si hay contenido oculto.*

### ¿SE PUEDE MODIFICAR EL ANCHO DE COLUMNAS O EL LÍMITE DE FILAS?

*No directamente vía parámetros públicos, pero se podría incorporar a posteriori un ajuste en el código fuente cambiando las constantes de configuración (por ejemplo, MAX\_FILAS\_A\_MOSTRAR).*

### ¿SE PUEDE EXPORTAR LA TABLA VISUALIZADA A UN ARCHIVO DE TEXTO?

*No directamente desde este paquete (Visualizador), pero podés redirigir la salida estándar de Java, esto significa que al ejecutar el programa que hace uso de la librería desde la consola, se podría agregar un parámetro al comando como "`> archivo_de_salida.txt`", por ejemplo ejecutando "`java Programa > salida.txt`". Entonces lo que normalmente se mostraría en pantalla quedaría guardado en un archivo*

*A posterior se podría adaptar el método de impresión, pero hoy no está disponible en esta librería. De manera tal que los métodos de impresión del paquete Visualizador, en vez de usar System.out, escriban directamente en un archivo.*

### ¿QUÉ PASA SI UNA CELDA TIENE MUCHO TEXTO?

*El contenido se trunca y se indica con "...", para evitar romper el formato de la tabla.*

### ¿VISUALIZADOR PERMITE FILTRAR O SELECCIONAR DATOS?

*No, solo muestra lo que recibe. Para seleccionar o filtrar datos, usá los módulos correspondientes antes de visualizar.*

### ¿CÓMO VEO EL TIPO DE DATO DE CADA COLUMNA?

*Usá el método Visualizador.info(tabla) para obtener un resumen de los tipos y la estructura.*



## PAQUETE: FILTROS

### DESCRIPCIÓN GENERAL

El paquete filtros proporciona una estructura para realizar selecciones avanzadas sobre los datos almacenados en una tabla. Permite aplicar condiciones simples (igualdad, desigualdad, comparación numérica y alfabética), combinarlas con operadores lógicos (AND, OR, NOT), y generar vistas parciales de las tablas originales con los resultados filtrados.

Este módulo ofrece una API mediante interfaces, clases concretas y una factoría de filtros para facilitar la creación y el uso de filtros personalizados.

### ENUMERACIONES FUNDAMENTALES

---

#### TIPOCOMPARADOR

Define los tipos básicos de comparadores disponibles para crear filtros simples:

- IGUAL (=)
- DISTINTO (!=)
- MAYOR (>)
- MENOR (<)
- MAYOR\_IGUAL (>=)
- MENOR\_IGUAL (<=)

---

#### OPERADORLOGICO

Define los operadores lógicos para combinar filtros múltiples:

- AND (&&): ambas condiciones deben cumplirse.
- OR (||): al menos una condición debe cumplirse.

---

#### TIPOSINPARAMETRO

Define filtros especiales que no requieren parámetros adicionales:

- ES\_NULO: filtra celdas cuyo valor es NA.

## INTERFACES PRINCIPALES

### FILTRO

Interfaz básica implementada por todos los filtros simples. Define el método principal:

- `boolean cumple(Object valorCelda, TipoDato tipoDatoCelda)`

### FILTROLOGICO

Interfaz para filtros compuestos que operan sobre dos columnas distintas en la misma fila:

- `boolean cumpleEnFila(Tabla tabla, Etiqueta etiquetaFila, Etiqueta etiquetaColumna1, Etiqueta etiquetaColumna2)`

### FILTROSINPARAMETRO

Interfaz para filtros que no requieren valores externos de comparación, operando directamente sobre una celda específica:

- `boolean cumpleEnFila(Tabla tabla, Etiqueta etiquetaFila, Etiqueta etiquetaColumna)`

## CLASES AUXILIARES Y FILTROS CONCRETOS (BÁSICOS)

Las siguientes clases implementan la interfaz Filtro para aplicar condiciones concretas y simples:

- **Igual:** verifica si el valor es exactamente igual al proporcionado.
- **Distinto:** verifica desigualdad.
- **Mayor:** comprueba si el valor es estrictamente mayor.
- **Menor:** comprueba si el valor es estrictamente menor.
- **MayorIgual:** comprueba si el valor es mayor o igual.
- **MenorIgual:** comprueba si el valor es menor o igual.

Además, está la clase especial:

- **EsNulo:** implementa FiltroSinParametro para seleccionar celdas cuyo valor es faltante (NA).

## EJEMPLO DE USO (FILTRADO SIMPLE)

```
List<Etiqueta> etiquetas = Arrays.asList(
    new EtiquetaString("Alumno"),
    new EtiquetaString("Nota")
);

List<TipoDato> tipos = Arrays.asList(
    TipoDato.CADENA,
    TipoDato.NUMERICO
);

Tabla tabla = new Tabla(etiquetas, tipos);
tabla.agregarFila(Arrays.asList(new Celda<>("Juan", TipoDato.CADENA), new Celda<>(7, TipoDato.NUMERICO)));
tabla.agregarFila(Arrays.asList(new Celda<>("Ana", TipoDato.CADENA), new Celda<>(9, TipoDato.NUMERICO)));
tabla.agregarFila(Arrays.asList(new Celda<>("Luis", TipoDato.CADENA), new Celda<>(5, TipoDato.NUMERICO)));

// filtrado simple (notas mayores o iguales a 7)
ControladorFiltro controlador = new ControladorFiltro(tabla);
controlador.filtroSimple(new EtiquetaString("Nota"), TipoComparador.MAYOR_IGUAL, 7);
```

## SALIDA POR CONSOLA

FILA	Alumno	Nota
0	Juan	7
1	Ana	9

## FAQ (FILTROS SIMPLES)

### ¿PUEDO APLICAR UN FILTRO A CUALQUIER TIPO DE DATO?

Sí, pero la comparación debe ser coherente (por ejemplo, numérico con numérico, cadena con cadena, etc.).

### ¿QUÉ SUCEDE SI INTENTO COMPARAR VALORES INCOMPATIBLES, POR EJEMPLO, UN TEXTO CON UN NÚMERO?

El filtro devolverá siempre false para esos casos y no seleccionará la fila.

### ¿SE PUEDE FILTRAR CELDAS VACÍAS?

Sí, usando el filtro especial `ES_NULO`.

### ¿ES POSIBLE COMBINAR VARIAS CONDICIONES DE FILTRADO?

Sí, usando filtros lógicos (AND, OR) o negación (NOT). Estos se documentan más adelante.

## FILTROS COMPUESTOS (LÓGICOS)

### FILTROAND

Permite combinar dos filtros simples o condiciones, seleccionando filas donde ambas condiciones se cumplan simultáneamente.

### EJEMPLO DE USO

Continuando con el caso anterior:

```
// Filtrado de alumnos con Nota >= 7 Y nombre alfabéticamente después de "H"
Filtro notaAlta = FiltroFactory.crearFiltroComparador(TipoComparador.MAYOR_IGUAL, 7);
Filtro nombreDespuesH = FiltroFactory.crearFiltroComparador(TipoComparador.MAYOR, "H");

FiltroLogico and = FiltroFactory.crearFiltroLogico(OperadorLogico.AND, notaAlta, nombreDespuesH);

ControladorFiltro controlador = new ControladorFiltro(tabla);
controlador.filtroCompuesto(OperadorLogico.AND,
    new EtiquetaString("Nota"), TipoComparador.MAYOR_IGUAL, 7,
    new EtiquetaString("Alumno"), TipoComparador.MAYOR, "H");
```

### SALIDA POR CONSOLA

FILA	Alumno	Nota
0	Juan	7

## FILTROOR

Combina dos condiciones, seleccionando filas que cumplan al menos una de ellas.

## EJEMPLO DE USO

Continuando con el caso anterior:

```
// Filtrado de alumnos con Nota < 6 O Nota > 8
Filtro notaBaja = FiltroFactory.crearFiltroComparador(TipoComparador.MENOR, 6);
Filtro notaAlta = FiltroFactory.crearFiltroComparador(TipoComparador.MAYOR, 8);

FiltroLogico or = FiltroFactory.crearFiltroLogico(OperadorLogico.OR, notaBaja, notaAlta);

ControladorFiltro controlador = new ControladorFiltro(tabla);
controlador.filtroCompuesto(OperadorLogico.OR,
    new EtiquetaString("Nota"), TipoComparador.MENOR, 6,
    new EtiquetaString("Nota"), TipoComparador.MAYOR, 8);
```

## SALIDA POR CONSOLA

FILA	Alumno	Nota
1	Ana	9
2	Luis	5

## FILTRONOT

Negación lógica de un filtro. Selecciona filas donde no se cumple la condición especificada.

## EJEMPLO DE USO

Continuando con el caso anterior:

```
// Filtrado de alumnos que NO tienen Nota = 7
Filtro filtroIgual7 = FiltroFactory.crearFiltroComparador(TipoComparador.IGUAL, 7);
FiltroNOT not = FiltroFactory.crearFiltroNOT(filtroIgual7);

ControladorFiltro controlador = new ControladorFiltro(tabla);
controlador.filtrarNOT(new EtiquetaString("Nota"), TipoComparador.IGUAL, 7);
```

## SALIDA POR CONSOLA

FILA	Alumno	Nota
1	Ana	9
2	Luis	5

## CLASES Y FUNCIONES AUXILIARES AVANZADAS

### FILTROFACTORY

Clase que utiliza el patrón Factory para facilitar la creación de filtros y combinarlos sin exponer al usuario la complejidad de las instanciaciones individuales.

### MÉTODOS PRINCIPALES

- crearFiltroComparador(TipoComparador tipo, Object valorAComparar)
- crearFiltroLogico(OperadorLogico operador, Filtro filtro1, Filtro filtro2)
- crearFiltroNOT(Filtro filtroInterno)
- crearFiltroSinParametros(TipoSinParametro tipoFiltro)

---

## TABLAPARCIAL

Genera una nueva tabla con una selección específica de filas y columnas originales, basada en los resultados del filtrado.

**Uso interno del paquete.**

---

## FILTRADO

Clase interna que implementa la lógica real de filtrado y devuelve tablas parciales según los filtros aplicados.

---

### MÉTODOS PRINCIPALES

- `filtrarComparador(...)`: Aplica filtros simples.
- `filtrarLogico(...)`: Aplica filtros compuestos (AND, OR).
- `filtrarNot(...)`: Aplica el filtro negado (NOT).
- `filtrarSinParametros(...)`: Aplica filtros sin parámetros (ej: ES\_NULO).

---

## CONTROLADORFILTRO

Clase pública que actúa como interfaz de usuario final, combinando todas las funcionalidades anteriores y simplificando el uso de los filtros.

---

### MÉTODOS PRINCIPALES

- `filtroSimple(...)`: Filtra por condición sencilla.
- `filtroCompuesto(...)`: Filtra combinando condiciones lógicas (AND, OR).
- `filtrarNOT(...)`: Filtra mediante negación de una condición.
- `filtrarSinParametros(...)`: Filtra condiciones sin parámetros adicionales (ej. celdas NA).

## EJEMPLO DE USO (FILTRADO COMPUESTO)

```
import gestiondedatos.*;
import filtros.*;
import java.util.*;

public class TestFiltradoCompleto {
    public static void main(String[] args) {
        List<Etiqueta> etiquetas = Arrays.asList(
            new EtiquetaString("Producto"),
            new EtiquetaString("Precio"),
            new EtiquetaString("Stock")
        );
        List<TipoDato> tipos = Arrays.asList(
            TipoDato.CADENA,
            TipoDato.NUMERICO,
            TipoDato.NUMERICO
        );
        Tabla tabla = new Tabla(etiquetas, tipos);

        tabla.agregarFila(Arrays.asList(new Celda<>("Yerba", TipoDato.CADENA), new Celda<>(500, TipoDato.NUMERICO), new Celda<>(50,
TipoDato.NUMERICO)));
        tabla.agregarFila(Arrays.asList(new Celda<>("Café", TipoDato.CADENA), new Celda<>(1500, TipoDato.NUMERICO), new Celda<>(0,
TipoDato.NUMERICO)));
        tabla.agregarFila(Arrays.asList(new Celda<>("Té", TipoDato.CADENA), new Celda<>(300, TipoDato.NUMERICO), new Celda<>(20,
TipoDato.NUMERICO)));

        ControladorFiltro controlador = new ControladorFiltro(tabla);

        // Filtrado de productos caros (precio >= 1000) O sin stock (stock = 0)
        controlador.filtroCompuesto(OperadorLogico.OR,
            new EtiquetaString("Precio"), TipoComparador.MAYOR_IGUAL, 1000,
            new EtiquetaString("Stock"), TipoComparador.IGUAL, 0);
    }
}
```

## SALIDA POR CONSOLA

FILA	Producto	Precio	Stock
1	Café	1500	0



## RELACIÓN CON EL RESTO DEL SISTEMA

- Los filtros interactúan con el núcleo gestionedatos, utilizando su estructura para realizar selecciones precisas.
- La visualización de resultados del filtrado se realiza mediante el paquete visualizador.

## RESUMEN DEL PAQUETE COMPLETO

- El paquete filtros permite selección avanzada y flexible de datos tabulares.
- Facilita combinaciones complejas de condiciones lógicas (AND, OR, NOT).
- Usa una factoría interna (FiltroFactory) que simplifica la creación y gestión de filtros.
- Las tablas filtradas se devuelven como vistas parciales (TablaParcial) para análisis posterior.

## FAQ (FILTROS AVANZADOS)

### ¿PUEDO COMBINAR MÁS DE DOS CONDICIONES A LA VEZ?

*Sí, encadenando múltiples filtros lógicos, aunque cada instancia lógica (AND, OR) sólo combina dos condiciones a la vez.*

### ¿QUÉ SUCEDE SI APLICO UN FILTRO Y NO QUEDAN FILAS?

*Obtendrás una tabla parcial vacía, sin errores ni excepciones.*

### ¿SE MODIFICAN MIS DATOS ORIGINALES AL FILTRAR?

*No, el filtrado devuelve vistas parciales independientes, dejando intacta la tabla original.*

### ¿SE PUEDEN AÑADIR FILTROS PERSONALIZADOS?

*No, para esto se debería modificar el código, implementando las interfaces Filtro, FiltroLogico o FiltroSinParametro y registrándolos en la factoría correspondiente.*

## PAQUETE: SELECCIÓN

### DESCRIPCIÓN GENERAL

El paquete seleccion permite obtener vistas parciales rápidas y sencillas de los datos contenidos en una tabla. Brinda métodos intuitivos para seleccionar las primeras o últimas filas, así como para seleccionar filas y columnas específicas mediante etiquetas. Es útil para realizar análisis preliminares rápidos o verificar visualmente los datos antes de operaciones más complejas.

### CLASE PRINCIPAL

#### SELECCION

La clase Seleccion se utiliza para realizar selecciones parciales sobre los datos de una tabla existente.

#### CONSTRUCTOR

Seleccion(Tabla tabla): Inicializa un objeto de selección asociado a una tabla específica.

### MÉTODOS PRINCIPALES

#### HEAD (INT NUMFILAS)

Muestra las primeras numFilas filas de la tabla.

- Parámetros: número entero positivo indicando cuántas filas mostrar desde el inicio.
- Uso típico: revisar rápidamente los primeros registros.

#### TAIL (INT NUMFILAS)

Muestra las últimas numFilas filas de la tabla.

- Parámetros: número entero positivo indicando cuántas filas mostrar desde el final.
- Uso típico: observar rápidamente los últimos registros cargados o generados.

## SELECCIONAR(LIST<ETIQUETA> ETIQUETAFILAS, LIST<ETIQUETA> ETIQUETASCOLUMNAS)

Muestra una vista parcial de la tabla original según etiquetas de filas y columnas específicas.

- **Parámetros:**
  - etiquetasFilas: lista de etiquetas (String o Entero) indicando las filas deseadas.
  - etiquetasColumnas: lista de etiquetas indicando las columnas deseadas.
- **Uso típico:** extraer conjuntos específicos de datos para inspección o análisis posterior.

### EJEMPLO DE USO

```
List<Etiqueta> etiquetas = Arrays.asList( new EtiquetaString("Ciudad"), new EtiquetaString("Habitantes"), new EtiquetaString("País"));
List<TipoDato> tipos = Arrays.asList( TipoDato.CADENA, TipoDato.NUMERICO, TipoDato.CADENA );

Tabla tabla = new Tabla(etiquetas, tipos);

tabla.agregarFila(Arrays.asList(new Celda<>("Madrid", TipoDato.CADENA), new Celda<>(3200000, TipoDato.NUMERICO), new
Celda<>("España", TipoDato.CADENA)));

tabla.agregarFila(Arrays.asList(new Celda<>("Roma", TipoDato.CADENA), new Celda<>(2800000, TipoDato.NUMERICO), new Celda<>("Italia",
TipoDato.CADENA)));

tabla.agregarFila(Arrays.asList(new Celda<>("París", TipoDato.CADENA), new Celda<>(2140000, TipoDato.NUMERICO), new
Celda<>("Francia", TipoDato.CADENA)));

tabla.agregarFila(Arrays.asList(new Celda<>("Berlín", TipoDato.CADENA), new Celda<>(3800000, TipoDato.NUMERICO), new
Celda<>("Alemania", TipoDato.CADENA)));

Seleccion seleccion = new Seleccion(tabla);

// mostramos las primeras 2 filas
seleccion.head(2);

// Mostramos las últimas 2 filas
seleccion.tail(2);

// hacemos una selección específica de filas y columnas
List<Etiqueta> filasSeleccionadas = Arrays.asList(new EtiquetaEntero(0), new EtiquetaEntero(2)); // Madrid y París
List<Etiqueta> columnasSeleccionadas = Arrays.asList(new EtiquetaString("Ciudad"), new EtiquetaString("País"));
seleccion.seleccionar(filasSeleccionadas, columnasSeleccionadas);
}
```

## SALIDA POR CONSOLA

FILA	Ciudad	Habitantes	País
0	Madrid	3200000	España
1	Roma	2800000	Italia

FILA	Ciudad	Habitantes	País
2	París	2140000	Francia
3	Berlín	3800000	Alemania

FILA	Ciudad	País
0	Madrid	España
2	París	Francia

## RELACIÓN CON EL RESTO DEL SISTEMA

- El paquete seleccion depende directamente de los paquetes gestionedatos y visualizador para acceder a los datos originales y mostrarlos.
- Utiliza la clase auxiliar TablaParcial del paquete filtros para generar vistas parciales sin alterar la tabla original.
- Es útil antes de operaciones de manipulación o filtrado para una rápida inspección visual.

## RESUMEN

- Permite seleccionar vistas rápidas de tablas por filas (head o tail) o selección personalizada de filas y columnas.
- Es ideal para inspecciones rápidas, preliminares o parciales de datos.
- Facilita el análisis visual sin afectar ni modificar la estructura original de los datos.

## FAQ

**¿QUÉ PASA SI PIDO MÁS FILAS DE LAS QUE TIENE LA TABLA CON HEAD() O TAIL()?**

*La operación arrojará una excepción explicativa indicando que el número solicitado excede las filas disponibles.*

**¿LA SELECCIÓN MODIFICA MI TABLA ORIGINAL?**

*No, se generan tablas parciales independientes que solo muestran el contenido seleccionado, dejando intacta la tabla original.*

**¿PUEDO SELECCIONAR ETIQUETAS QUE NO EXISTEN EN LA TABLA ORIGINAL?**

*No. Si seleccionás etiquetas inexistentes, recibirás una excepción informativa indicando claramente qué etiquetas no son válidas.*

**¿SE PUEDEN SELECCIONAR COLUMNAS O FILAS CON ETIQUETAS NUMÉRICAS Y TEXTUALES SIMULTÁNEAMENTE?**

*Sí, siempre y cuando las etiquetas existan en la tabla original.*

## PAQUETE: COPIAYCONCATENACION

### DESCRIPCIÓN GENERAL

El paquete `copyayconcatenacion` ofrece utilidades para crear copias profundas de tablas existentes y concatenar múltiples tablas en una sola estructura. Garantiza que los datos copiados sean totalmente independientes de los originales, evitando efectos colaterales. Además, facilita la unión vertical (por filas) de varias tablas, asegurando previamente que sean compatibles en estructura.

### CLASES PRINCIPALES

#### COPIATABLA

Permite generar una copia profunda e independiente de una tabla existente. Se asegura de copiar no solo la estructura sino también cada celda de manera individual.

---

#### MÉTODO PRINCIPAL

- `static Tabla copiarTabla(Tabla tablaOriginal)`

---

#### CONCATENACIONTABLA

Permite unir dos tablas verticalmente, verificando previamente que tengan la misma estructura en términos de cantidad de columnas, tipos de datos y etiquetas.

---

#### MÉTODO PRINCIPAL

- `static Tabla concatenarTablas(Tabla tabla1, Tabla tabla2)`

## EJEMPLO DE USO

```

List<Etiqueta> etiquetas = Arrays.asList( new EtiquetaString("Nombre"), new EtiquetaString("Edad") );
List<TipoDato> tipos = Arrays.asList( TipoDato.CADENA, TipoDato.NUMERICO );

// Creamos tabla original
Tabla tablaOriginal = new Tabla(etiquetas, tipos);
tablaOriginal.agregarFila(Arrays.asList(new Celda<>("Carlos", TipoDato.CADENA), new Celda<>(25, TipoDato.NUMERICO)));
tablaOriginal.agregarFila(Arrays.asList(new Celda<>("María", TipoDato.CADENA), new Celda<>(30, TipoDato.NUMERICO)));

// Copiamos tabla original
Tabla tablaCopia = CopiaTabla.copiarTabla(tablaOriginal);
Visualizador.imprimirTabla(tablaCopia);

// Creamos otra tabla compatible para concatenación
Tabla otraTabla = new Tabla(etiquetas, tipos);
otraTabla.agregarFila(Arrays.asList(new Celda<>("Ana", TipoDato.CADENA), new Celda<>(22, TipoDato.NUMERICO)));
otraTabla.agregarFila(Arrays.asList(new Celda<>("Luis", TipoDato.CADENA), new Celda<>(28, TipoDato.NUMERICO)));

// Concatenamos las dos tablas
Tabla tablaConcatenada = ConcatenacionTabla.concatenarTablas(tablaOriginal, otraTabla);
Visualizador.imprimirTabla(tablaConcatenada);

```

## SALIDA POR CONSOLA

FILA	Nombre	Edad
0	Carlos	25
1	María	30

FILA	Nombre	Edad
0	Carlos	25
1	María	30
2	Ana	22
3	Luis	28

## RELACIÓN CON EL RESTO DEL SISTEMA

- Este paquete interactúa muy estrechamente con el paquete core gestiondedatos, usando sus estructuras (tabla, columna, celda) como base para copias y concatenaciones.
- Utiliza las excepciones definidas en gestiondeerrores para informar sobre inconsistencias o errores estructurales.
- Los resultados suelen mostrarse mediante el paquete visualizador.

## RESUMEN

- Permite realizar copias independientes de tablas existentes para evitar modificaciones accidentales en datos originales.
- Facilita la combinación vertical de tablas que compartan una misma estructura y tipos.
- Verifica la compatibilidad antes de realizar operaciones para asegurar integridad.

## FAQ

### ¿QUÉ OCURRE SI INTENTO CONCATENAR TABLAS QUE NO COINCIDEN EXACTAMENTE EN SU ESTRUCTURA?

*Recibirás una excepción informativa detallando claramente cuál es la incompatibilidad (etiquetas, cantidad de columnas o tipos de dato).*

### ¿LAS COPIAS REALIZADAS SON TOTALMENTE INDEPENDIENTES?

*Sí, se generan copias profundas. Cualquier modificación sobre la copia no afecta a la tabla original ni viceversa.*

### ¿PUEDO CONCATENAR MÁS DE DOS TABLAS A LA VEZ?

*Actualmente, solo se permite concatenar dos tablas simultáneamente. Podés concatenar múltiples tablas realizando concatenaciones sucesivas.*

### ¿QUÉ PASA CON LAS ETIQUETAS DE FILAS AL CONCATENAR TABLAS?

*Las etiquetas textuales (EtiquetaString) se copian tal cual. Las etiquetas numéricas (EtiquetaEntero) se regeneran para asegurar coherencia e integridad en la numeración secuencial.*



## PAQUETE: MANIPULACION

### DESCRIPCIÓN GENERAL

Este paquete proporciona utilidades para transformar o modificar los datos contenidos en una Tabla, enfocándose en tres operaciones comunes de preprocesamiento: ordenamiento, muestreo aleatorio e imputación de valores faltantes. Todas las clases implementan una interfaz común ManipuladorDatos, lo que permite aplicar los manipuladores de forma uniforme sobre diferentes tablas.

### CLASES PRINCIPALES

---

#### ORDENADORDATOS

La clase OrdenadorDatos permite ordenar las filas de una tabla según el valor de una o más columnas, en orden ascendente (por defecto).

---

#### CONSTRUCTOR

- `OrdenadorDatos(List<String> columnasOrden)`: Recibe una lista con los nombres de las columnas por las que se quiere ordenar la tabla (la primera columna tiene mayor prioridad).

---

#### MÉTODO PRINCIPAL

- `manipular(Tabla tabla)`: Ordena la tabla “in place” (es decir, modifica el orden de las filas en la estructura original) según los criterios definidos al instanciar el objeto.

## EJEMPLO DE USO

```

//armamos la tabla con datos desordenados

Tabla tabla = new Tabla();

EtiquetaString etColEdad = new EtiquetaString("Edad");
EtiquetaString etColNombre = new EtiquetaString("Nombre");

Columna colEdad = new Columna(TipoDato.NUMERICO);
Columna colNombre = new Columna(TipoDato.CADENA);

//usamos listas para asegurar alineación Edad-Nombre
List<Integer> edades = Arrays.asList(22, 34, 20, 30);
List<String> nombres = Arrays.asList("Ludmi", "Lean", "Santi", "Mica");

for (int i = 0; i < edades.size(); i++) {
    colEdad.agregarCelda(new Celda<>(edades.get(i), TipoDato.NUMERICO));
    colNombre.agregarCelda(new Celda<>(nombres.get(i), TipoDato.CADENA));
}
tabla.agregarColumna(etColEdad, colEdad);
tabla.agregarColumna(etColNombre, colNombre);
tabla.generarEtiquetasFilas();

//mostramos antes de ordenar
System.out.println("Tabla antes de ordenar:");
for (Etiqueta etiquetaFila : tabla.getEtiquetasFilas()) {
    Integer edad = (Integer) tabla.getCelda(etColEdad, etiquetaFila).getValor();
    String nombre = (String) tabla.getCelda(etColNombre, etiquetaFila).getValor();
    System.out.println("Edad: " + edad + ", Nombre: " + nombre);
}

//ordenamos por Edad y medimos eficiencia
OrdenadorDatos ordenador = new OrdenadorDatos(Collections.singletonList("Edad"));
MedidorEficiencia.medirTiempo("ordenamiento por Edad", () -> {
    ordenador.manipular(tabla);
});

//mostramos después de ordenar
System.out.println("\nTabla después de ordenar por Edad (de menor a mayor):");
for (int i = 0; i < tabla.contarFilas(); i++) {
    Etiqueta etiquetaFila = tabla.getEtiquetasFilas().get(i);
    Integer edad = (Integer) tabla.getCelda(etColEdad, etiquetaFila).getValor();
    String nombre = (String) tabla.getCelda(etColNombre, etiquetaFila).getValor();
    System.out.println("Edad: " + edad + ", Nombre: " + nombre + " (Etiqueta: " + etiquetaFila.getValor() + ")");
}

System.out.println("\nComprobación de valores post-sort por índice (no por etiqueta):");
for (int i = 0; i < tabla.contarFilas(); i++) {
    Integer edad = (Integer) colEdad.getCelda(i).getValor();
    String nombre = (String) colNombre.getCelda(i).getValor();
    System.out.println("Índice: " + i + " Edad: " + edad + ", Nombre: " + nombre);
}

```

---

## SALIDA POR CONSOLA

```
Tabla antes de ordenar:
Edad: 22, Nombre: Ludmi
Edad: 34, Nombre: Lean
Edad: 20, Nombre: Santi
Edad: 30, Nombre: Mica
Tiempo de ordenamiento por Edad: 0.0041947 segundos

Tabla después de ordenar por Edad (de menor a mayor):
Edad: 20, Nombre: Santi (Etiqueta: 2)
Edad: 22, Nombre: Ludmi (Etiqueta: 0)
Edad: 30, Nombre: Mica (Etiqueta: 3)
Edad: 34, Nombre: Lean (Etiqueta: 1)

Comprobación de valores post-sort por índice (no por etiqueta):
Índice: 0 Edad: 20, Nombre: Santi
Índice: 1 Edad: 22, Nombre: Ludmi
Índice: 2 Edad: 30, Nombre: Mica
Índice: 3 Edad: 34, Nombre: Lean
```

## MUESTREADORDATOS

La clase MuestreadorDatos permite seleccionar aleatoriamente un porcentaje de las filas de la tabla, eliminando el resto.

---

## CONSTRUCTOR

MuestreadorDatos(double porcentaje): Recibe el porcentaje de filas a conservar (por ejemplo, 0.5 para quedarse con el 50%).

---

## MÉTODO PRINCIPAL

manipular(Tabla tabla): Aplica el muestreo, dejando solo la cantidad de filas seleccionada al azar.

## EJEMPLO DE USO

```
//creamos una tabla con 6 filas
Tabla tabla = new Tabla();
EtiquetaString etColId = new EtiquetaString("ID");
EtiquetaString etColNombre = new EtiquetaString("Nombre");
Columna colId = new Columna(TipoDato.NUMERICO);
Columna colNombre = new Columna(TipoDato.CADENA);
for (int i = 1; i <= 6; i++) {
    colId.agregarCelda(new Celda<>(i, TipoDato.NUMERICO));
    colNombre.agregarCelda(new Celda<>("Persona" + i, TipoDato.CADENA));
}
tabla.agregarColumna(etColId, colId);
tabla.agregarColumna(etColNombre, colNombre);
tabla.generarEtiquetasFilas();
//mostramos antes de muestrear
System.out.println("Tabla original:");
for (int i = 0; i < tabla.contarFilas(); i++) {
    System.out.print("Fila " + i + ": ");
    System.out.print(tabla.getCelda(etColId, tabla.getEtiquetasFilas().get(i)) + " - ");
    System.out.println(tabla.getCelda(etColNombre, tabla.getEtiquetasFilas().get(i)));
}
//muestreo (quedarse con el 50%)
double porcentaje = 0.5;
MuestreadorDatos muestreador = new MuestreadorDatos(porcentaje);
muestreador.manipular(tabla);

//mostramos después de muestrear
System.out.println("\nTabla después de muestrear (" + (int)(porcentaje * 100) + "% de las filas:");
for (int i = 0; i < tabla.contarFilas(); i++) {
    System.out.print("Fila " + i + ": ");
    System.out.print(tabla.getCelda(etColId, tabla.getEtiquetasFilas().get(i)) + " - ");
    System.out.println(tabla.getCelda(etColNombre, tabla.getEtiquetasFilas().get(i)));
}

System.out.println("\nCantidad final de filas: " + tabla.contarFilas());
```

---

## SALIDA POR CONSOLA

```
Tabla original:
Fila 0: 1 - Persona1
Fila 1: 2 - Persona2
Fila 2: 3 - Persona3
Fila 3: 4 - Persona4
Fila 4: 5 - Persona5
Fila 5: 6 - Persona6

Tabla después de muestrear (50% de las filas):
Fila 0: 1 - Persona1
Fila 1: 4 - Persona4
Fila 2: 6 - Persona6

Cantidad final de filas: 3
```

## IMPUTADORDATOS

La clase `ImputadorDatos` permite rellenar todas las celdas con valor faltante (NA) por un valor literal definido por el usuario.

---

### CONSTRUCTOR

- `ImputadorDatos(Object valorImputacion)`: Define el valor a utilizar para reemplazar los NA.

---

### MÉTODO PRINCIPAL

- `manipular(Tabla tabla)`: Reemplaza todos los valores NA por el valor de imputación, respetando el tipo de dato de cada columna.

## EJEMPLO DE USO

```
//creamos una tabla con dos columnas, una con algunos valores NA

Tabla tabla = new Tabla();

EtiquetaString etColNum = new EtiquetaString("Puntaje");
EtiquetaString etColStr = new EtiquetaString("Nombre");

Columna colNum = new Columna(TipoDato.NUMERICO);
colNum.agregarCelda(new Celda<>(10, TipoDato.NUMERICO));
colNum.agregarCelda(new Celda<>());
colNum.agregarCelda(new Celda<>(7, TipoDato.NUMERICO));
colNum.agregarCelda(new Celda<>());

Columna colStr = new Columna(TipoDato.CADENA);
colStr.agregarCelda(new Celda<>("Ana", TipoDato.CADENA));
colStr.agregarCelda(new Celda<>("Luis", TipoDato.CADENA));
colStr.agregarCelda(new Celda<>());
colStr.agregarCelda(new Celda<>());

tabla.agregarColumna(etColNum, colNum);
tabla.agregarColumna(etColStr, colStr);
tabla.generarEtiquetasFilas();

//mostramos tabla antes de imputar
System.out.println("Tabla antes de imputar:");
for (int i = 0; i < tabla.contarFilas(); i++) {
    System.out.print("Fila " + i + ": ");
    System.out.print(tabla.getCelda(etColNum, tabla.getEtiquetasFilas().get(i)) + " - ");
    System.out.println(tabla.getCelda(etColStr, tabla.getEtiquetasFilas().get(i)));
}

//imputamos NA de Puntaje con 0
ImputadorDatos imputadorNum = new ImputadorDatos(0);
imputadorNum.manipular(tabla);

//imputamos NA de Nombre con "Desconocido"
ImputadorDatos imputadorStr = new ImputadorDatos("Desconocido");
imputadorStr.manipular(tabla);

//mostramos tabla después de imputar
System.out.println("\nTabla después de imputar:");
for (int i = 0; i < tabla.contarFilas(); i++) {
    System.out.print("Fila " + i + ": ");
    System.out.print(tabla.getCelda(etColNum, tabla.getEtiquetasFilas().get(i)) + " - ");
    System.out.println(tabla.getCelda(etColStr, tabla.getEtiquetasFilas().get(i)));
}
```

## SALIDA POR CONSOLA

```
Tabla antes de imputar:  
Fila 0: 10 - Ana  
Fila 1: NA - Luis  
Fila 2: 7 - NA  
Fila 3: NA - NA  
  
Tabla después de imputar:  
Fila 0: 10 - Ana  
Fila 1: 0 - Luis  
Fila 2: 7 - Desconocido  
Fila 3: 0 - Desconocido
```

## RELACIÓN CON EL RESTO DEL SISTEMA

- El paquete manipulación utiliza directamente las clases de gestión de datos (Tabla, Columna, Celda) y opera siempre sobre la estructura interna de la tabla.
- Las operaciones pueden aplicarse luego de una selección, filtrado o limpieza previa.
- Es muy útil antes de visualizar datos o exportarlos, para asegurarse de que la tabla esté ordenada, limpia y lista para análisis.

## RESUMEN

- Permite ordenar, muestrear o imputar valores faltantes en una tabla, facilitando el preprocesamiento y la limpieza de datos.
- Opera sobre la estructura original de la tabla.
- Es fundamental para preparar los datos antes de análisis estadístico, visualización o exportación.

## FAQ

**¿LOS CAMBIOS AFECTAN LA TABLA ORIGINAL O GENERAN UNA NUEVA?**

*Todos los métodos de este paquete modifican la tabla original. Si necesitás mantener el original intacto, primero hacé una copia profunda.*

**¿QUÉ PASA SI IMPUTO UN VALOR DE TIPO INCORRECTO EN UNA COLUMNA?**

*El sistema validará el tipo antes de asignarlo. Si el valor de imputación no es compatible con el tipo de la columna, arrojará una excepción.*

**¿EL ORDENAMIENTO ADMITE VARIOS CRITERIOS?**

*Sí, podés ordenar por varias columnas, la lista que pasás al constructor define el orden de prioridad.*

**¿EL MUESTREO SIEMPRE DA EL MISMO RESULTADO?**

*No, el muestreo es aleatorio. Cada vez que lo ejecutás sobre la misma tabla, podés obtener filas distintas.*

**¿PUEDO IMPUTAR VALORES EN COLUMNAS DE DISTINTOS TIPOS SIMULTÁNEAMENTE?**

*No directamente en una sola operación, pero podés aplicar varias veces `ImputadorDatos` sobre cada columna, usando el valor adecuado para cada tipo.*

**¿QUÉ PASA SI HAY POCAS FILAS Y PIDO UN PORCENTAJE MUY BAJO EN EL MUESTREO?**

*Si el porcentaje resulta en menos de una fila, no se seleccionará ninguna (o sólo una, según la implementación), siempre respetando el mínimo posible.*



## PAQUETE: GESTION DE ERRORES

### DESCRIPCIÓN GENERAL

El paquete `gestiondeerrores` centraliza el manejo y la comunicación de errores y advertencias en toda la librería. Define una jerarquía de excepciones específicas y provee utilidades para registrar mensajes de error y advertencia, facilitando la detección y el diagnóstico de problemas durante la manipulación de datos tabulares.

### CLASES PRINCIPALES

#### GESTIONERRORES

Clase utilitaria con métodos estáticos para imprimir mensajes de error y advertencia de manera uniforme en la consola, permitiendo distinguir claramente los problemas graves de las situaciones menos críticas.

#### MÉTODOS PRINCIPALES

- `logError(Exception ex)`: Muestra un mensaje de error basado en una excepción.
- `logError(String mensaje)`: Muestra un mensaje de error personalizado.
- `logAdvertencia(String mensaje)`: Muestra un mensaje de advertencia.

#### EXCEPCIONTABULAR (ABSTRACTA)

Clase base para todas las excepciones específicas de la librería. Extiende `RuntimeException` y agrega la posibilidad de acceder al mensaje personalizado.

#### EXCEPCIONVALIDACION

Se lanza cuando ocurre un error de validación de datos, por ejemplo: intento de insertar un tipo de dato incompatible, o un valor no permitido.

#### EXCEPCIONOPERACIONNOVALIDA

Se utiliza cuando se intenta ejecutar una operación no permitida sobre la estructura de datos, como eliminar una columna inexistente o acceder a índices fuera de rango.

## EXCEPCIONCARGADATOS

Se lanza ante errores durante la carga o el parseo de datos desde archivos externos (por ejemplo, problemas de formato, archivos inexistentes, etc).

## NOTA

Este paquete es de uso interno. El usuario final no interactúa directamente con estas clases ni debe invocarlas en su propio código. Los mensajes y excepciones generados por estas clases se reflejan automáticamente al ocurrir errores en las operaciones sobre tablas, columnas o celdas.

## EJEMPLO DE USO

(No aplica. Este paquete es utilizado internamente por la librería para el manejo y comunicación de errores. El usuario no lo emplea de manera directa.)

## RELACIÓN CON EL RESTO DEL SISTEMA

- Es utilizado internamente por todos los módulos para lanzar, capturar y registrar errores o advertencias relevantes.
- Garantiza que los problemas se comuniquen de forma clara y homogénea, facilitando el diagnóstico y la robustez general del sistema.

## RESUMEN

- Centraliza y estandariza el manejo de errores para toda la librería.
- No requiere intervención ni conocimiento específico por parte del usuario final.
- Hace que los mensajes de error sean claros y consistentes en toda la librería.

## FAQ

### ¿PUEDO CAPTURAR ESTAS EXCEPCIONES EN MI CÓDIGO?

*Sí, si estás usando la librería en un proyecto Java propio, podés capturar las excepciones de tipo `ExcepcionTabular` (o sus hijas) para manejar errores personalizados. Sin embargo, no es necesario ni habitual invocar directamente las clases de este paquete.*

### ¿PUEDO MODIFICAR LOS MENSAJES DE ERROR?

*No directamente desde el código de usuario, pero los mensajes que aparecen en consola reflejan el origen y la causa del error de manera informativa.*

### ¿DEBO IMPORTAR ESTE PAQUETE PARA USAR LA LIBRERÍA?

*No, el paquete se usa internamente y no es necesario importarlo para manipular tablas o datos.*

## PAQUETE: HERRAMIENTAS

### DESCRIPCIÓN GENERAL

El paquete herramientas incluye utilidades auxiliares de propósito general para la medición y análisis de eficiencia en operaciones de la librería. Su principal objetivo es facilitar la evaluación del tiempo de ejecución de distintas tareas, lo cual resulta útil especialmente durante el desarrollo, pruebas y comparación de métodos de manipulación de datos.

### CLASE PRINCIPAL

#### MEDIDOREFICIENCIA

Esta clase proporciona métodos estáticos para medir y mostrar el tiempo de ejecución de bloques de código.

#### MÉTODOS PRINCIPALES

##### MEDIRTIEMPO(String DESCRIPCION, Runnable TAREA)

- Ejecuta una tarea (bloque de código) y muestra por consola cuánto tarda en completarse, junto a la descripción pasada como argumento.
- Uso típico: Evaluar la eficiencia de algoritmos de ordenamiento, muestreo, imputación, etc.

##### MARCARINICIO()

- Devuelve una marca de tiempo (en milisegundos) para registrar el inicio de un proceso.
- Uso típico: Medición manual y segmentada del tiempo de diferentes partes de un programa.

##### MOSTRARTIEMPOTOTAL(Long INICIO):

- Calcula y muestra el tiempo transcurrido desde una marca inicial dada (en milisegundos).
- Uso típico: Mostrar el tiempo total de ejecución de un programa o bloque grande de código.

## RELACIÓN CON EL RESTO DEL SISTEMA

- El paquete de herramientas se utiliza principalmente en tests y validaciones internas para comparar el desempeño de algoritmos o procesos.
- No afecta ni interviene en la lógica de manipulación de datos; su uso es opcional y enfocado en desarrolladores o testers.
- Puede ser empleada por cualquier módulo de la librería o por programas de usuario que quieran medir la eficiencia de ciertas operaciones.

## RESUMEN

- Permite medir de forma sencilla el tiempo de ejecución de operaciones o bloques de código.
- Ayuda a comparar y optimizar métodos, especialmente durante el desarrollo o evaluación de rendimiento.
- Es totalmente auxiliar y no impacta en el funcionamiento o estructura de las tablas ni de los datos.

## FAQ

### ¿ES OBLIGATORIO USAR ESTA CLASE AL UTILIZAR LA LIBRERÍA?

*No. Es un recurso opcional, pensado para desarrollo y pruebas de eficiencia.*

### ¿AFECTA EL FUNCIONAMIENTO DE LA TABLA SI NO USO HERRAMIENTAS?

*No, su uso es independiente y no modifica el comportamiento de los datos ni de la librería.*

### ¿PUEDO USAR MEDIDOREFICIENCIA PARA MEDIR CUALQUIER TIPO DE OPERACIÓN?

*Sí, cualquier bloque de código ejecutable.*

### ¿SE IMPRIME EL TIEMPO EN SEGUNDOS O MILLISEGUNDOS?

*Los resultados se muestran en segundos (con decimales para mayor precisión).*

## PAQUETE: AGREGACION

### DESCRIPCIÓN GENERAL

El paquete agregacion permite realizar operaciones de resumen y agrupamiento sobre tablas de datos. Proporciona mecanismos para agrupar filas según valores comunes en una o varias columnas y calcular métricas de agregación (suma, promedio, máximo, conteo, desvío estándar, etc.) sobre los grupos formados. Es el módulo clave para la generación de reportes, estadísticas o análisis exploratorios sintéticos a partir de los datos tabulares.

### PRINCIPALES CLASES Y COMPONENTES

#### TIPOOPERACION (ENUM)

Define los tipos de operaciones de agregación soportadas:

SUMA, MAXIMO, MINIMO, CUENTA, MEDIA, VARIANZA, DESVIO\_ESTANDAR

#### AGREGADORTABLA

Clase principal encargada de realizar la agrupación (groupBy) y la sumarización (summarize) de los datos.

- `groupBy(Tabla tabla, List<Etiqueta> columnasAgrupacion)`: agrupa la tabla según una o más columnas.
- `summarize(TablaAgrupada tablaAgrupada, List<Etiqueta> columnasASumarizar, TipoOperacion tipoOperacion)`: realiza la operación de resumen seleccionada sobre las columnas numéricas de cada grupo, devolviendo una nueva tabla con los resultados.

#### TABLAAGRUPADA

Clase que representa el resultado de agrupar una tabla.

- Almacena el mapa de grupos (clave: combinación de valores, valor: filas del grupo), las etiquetas de agrupación y la tabla original.

#### SUMARIZADOR (INTERFACE)

Define el contrato para todas las operaciones de agregación.

- Método: `Celda<?> sumarizar(List<Celda<?>> celdas)`

---

## IMPLEMENTACIONES DE SUMARIZADOR

- OperacionSuma: suma los valores numéricos.
- OperacionMaximo: máximo valor numérico.
- OperacionMinimo: mínimo valor numérico.
- OperacionCuenta: cantidad de valores no nulos/NA.
- OperacionMedia: promedio de valores numéricos.
- OperacionVarianza: varianza muestral.
- OperacionDesvioEstandar: desvío estándar (raíz de la varianza).

---

## FABRICAOPERACIONESSUMARIZACION

Permite obtener la estrategia adecuada de Sumarizador según el tipo de operación requerida.

---

## MÉTODOS PRINCIPALES

- `groupBy`: agrupa filas según el/los valores de las columnas seleccionadas. Devuelve una `TablaAgrupada`.
- `summarize`: aplica la operación agregada (ej. suma, promedio) sobre las columnas numéricas de cada grupo.

## EJEMPLO DE USO

```

//creamos etiquetas para las columnas
EtiquetaString etCurso = new EtiquetaString("Curso");
EtiquetaString etNota = new EtiquetaString("Nota");
EtiquetaString etEdad = new EtiquetaString("Edad");

//creamos las columnas correspondientes
Columna colCurso = new Columna(TipoDato.CADENA);
Columna colNota = new Columna(TipoDato.NUMERICO);
Columna colEdad = new Columna(TipoDato.NUMERICO);

// cargamos los datos: 6 alumnos de distintos cursos
// Curso, Nota, Edad
Object[][] datos = {
    {"1A", 8.0, 18}, {"1A", 6.5, 19}, {"2B", 7.0, 21}, {"2B", 6.6, 20}, {"1A", 7.2, 20},
    {"2B", 6.8, 22} };

for (Object[] fila : datos) {
    colCurso.agregarCelda(new Celda<>(fila[0], TipoDato.CADENA));
    colNota.agregarCelda(new Celda<>(fila[1], TipoDato.NUMERICO));
    colEdad.agregarCelda(new Celda<>(fila[2], TipoDato.NUMERICO));
}

// armamos la tabla
Tabla tabla = new Tabla();
tabla.agregarColumna(etCurso, colCurso);
tabla.agregarColumna(etNota, colNota);
tabla.agregarColumna(etEdad, colEdad);

// generamos las etiquetas de filas (numéricas por defecto)
tabla.generarEtiquetasFilas();

// mostramos la tabla original
System.out.println("Tabla original:");
Visualizador.imprimirTabla(tabla);

// agrupamos por curso y calculamos la nota promedio por curso
AgregadorTabla agregador = new AgregadorTabla();
TablaAgrupada agrupada = agregador.groupBy(tabla, Arrays.asList(etCurso));
Tabla tablaResumen = agregador.summarize(agrupada, Arrays.asList(etNota), TipoOperacion.MEDIA);

// Mostramos la tabla resumen
System.out.println("Promedio de nota por curso:");
Visualizador.imprimirTabla(tablaResumen);
}

```

## SALIDA POR CONSOLA

```
Tabla original:
FILA    Curso    Nota    Edad
0       1A       8.0     18
1       1A       6.5     19
2       2B       7.0     21
3       2B       6.6     20
4       1A       7.2     20
5       2B       6.8     22

Promedio de nota por curso:
FILA    Curso    Nota
0       2B       6.8
1       1A       7.23
```

## RELACIÓN CON EL RESTO DEL SISTEMA

- Este paquete depende de `gestiondedatos` para el manejo de tablas, filas y celdas.
- Se integra con el paquete `visualizador` para mostrar resultados de agrupaciones y resúmenes.
- Puede ser usado luego de una selección o filtrado previo para obtener estadísticas de subconjuntos de datos.
- Es fundamental para cualquier operación de reporting, síntesis y análisis tabular.

## RESUMEN

- Permite agrupar datos por una o varias columnas y calcular métricas resumidas sobre los grupos formados.
- Soporta operaciones clásicas de agregación: suma, promedio, máximo, mínimo, conteo, varianza, desvío estándar.
- Es esencial para la generación de reportes y análisis estadístico básico sobre datos.



## FAQ

**¿PUEDO AGRUPAR POR VARIAS COLUMNAS SIMULTÁNEAMENTE?**

*Sí, solo pasá la lista de etiquetas de columnas a `groupBy`.*

**¿QUÉ PASA SI AGRUPO POR UNA COLUMNA Y RESUMO OTRA NO NUMÉRICA?**

*Solo podés resumir columnas numéricas. Si intentás resumir columnas de otro tipo, se arroja una excepción.*

**¿PUEDO CALCULAR MÁS DE UNA MÉTRICA A LA VEZ (EJ: SUMA Y PROMEDIO)?**

*No directamente en una única llamada, pero podés llamar a `summarize` varias veces (una por operación) y combinar los resultados según tus necesidades.*

**¿CÓMO SE MANEJAN LOS VALORES NA O NULOS EN LAS OPERACIONES?**

*Son ignorados en los cálculos, salvo para el conteo, que cuenta solo valores válidos.*

**¿LOS GRUPOS SE CREAN AUNQUE HAYA COLUMNAS CON VALORES REPETIDOS O NULOS?**

*Sí, cada combinación única de valores de las columnas de agrupación genera un grupo. Los NA se tratan como un valor más para agrupar.*

## NOTAS FINALES

Este proyecto integrador fue realizado como parte de la cursada de Algoritmos I en la Universidad Nacional de San Martín. La librería implementa de manera modular las operaciones fundamentales sobre datos tabulares, permitiendo su reutilización y extensión. Quedan abiertas posibles mejoras, como la incorporación de nuevas operaciones estadísticas, mayor robustez en la gestión de errores y soporte para otros formatos de datos.

## CRÉDITOS

Desarrollado por:

LEANDRO VILLANUEVA

LUDMILA CÁCERES

MICAELA FLORIDIA

SANTINO SILVETTI

Materia: Algoritmos I – UNSAM

Docentes:

FABIANA TABOADA

FLORENCIA ROSSI

MIGUEL CARBONI

## AGRADECIMIENTOS

Agradecemos especialmente a los docentes de la materia, cuyo acompañamiento, orientación y dedicación fueron fundamentales para el desarrollo y finalización de este proyecto.