| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/DefaultRowSorter.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/swing/DefaultListSelectionModel.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/DefaultRowSorter.ModelWrapper.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/DefaultRowSorter.html)    [**NO FRAMES**](http://docs.google.com/DefaultRowSorter.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#3znysh7) | FIELD | [CONSTR](#tyjcwt) | [METHOD](#3dy6vkm) | DETAIL: FIELD | [CONSTR](#2s8eyo1) | [METHOD](#3rdcrjn) |

## **javax.swing**

Class DefaultRowSorter<M,I>

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 [javax.swing.RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<M>  
 **javax.swing.DefaultRowSorter<M,I>**

**Type Parameters:**M - the type of the modelI - the type of the identifier passed to the RowFilter **Direct Known Subclasses:** [TableRowSorter](http://docs.google.com/javax/swing/table/TableRowSorter.html)

public abstract class **DefaultRowSorter<M,I>**extends [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<M>

An implementation of RowSorter that provides sorting and filtering around a grid-based data model. Beyond creating and installing a RowSorter, you very rarely need to interact with one directly. Refer to [TableRowSorter](http://docs.google.com/javax/swing/table/TableRowSorter.html) for a concrete implementation of RowSorter for JTable.

Sorting is done based on the current SortKeys, in order. If two objects are equal (the Comparator for the column returns 0) the next SortKey is used. If no SortKeys remain or the order is UNSORTED, then the order of the rows in the model is used.

Sorting of each column is done by way of a Comparator that you can specify using the setComparator method. If a Comparator has not been specified, the Comparator returned by Collator.getInstance() is used on the results of calling toString on the underlying objects. The Comparator is never passed null. A null value is treated as occuring before a non-null value, and two null values are considered equal.

If you specify a Comparator that casts its argument to a type other than that provided by the model, a ClassCastException will be thrown when the data is sorted.

In addition to sorting, DefaultRowSorter provides the ability to filter rows. Filtering is done by way of a RowFilter that is specified using the setRowFilter method. If no filter has been specified all rows are included.

By default, rows are in unsorted order (the same as the model) and every column is sortable. The default Comparators are documented in the subclasses (for example, [TableRowSorter](http://docs.google.com/javax/swing/table/TableRowSorter.html)).

If the underlying model structure changes (the modelStructureChanged method is invoked) the following are reset to their default values: Comparators by column, current sort order, and whether each column is sortable. To find the default Comparators, see the concrete implementation (for example, [TableRowSorter](http://docs.google.com/javax/swing/table/TableRowSorter.html)). The default sort order is unsorted (the same as the model), and columns are sortable by default.

If the underlying model structure changes (the modelStructureChanged method is invoked) the following are reset to their default values: Comparators by column, current sort order and whether a column is sortable.

DefaultRowSorter is an abstract class. Concrete subclasses must provide access to the underlying data by invoking setModelWrapper. The setModelWrapper method **must** be invoked soon after the constructor is called, ideally from within the subclass's constructor. Undefined behavior will result if you use a DefaultRowSorter without specifying a ModelWrapper.

DefaultRowSorter has two formal type parameters. The first type parameter corresponds to the class of the model, for example DefaultTableModel. The second type parameter corresponds to the class of the identifier passed to the RowFilter. Refer to TableRowSorter and RowFilter for more details on the type parameters.

**Since:** 1.6 **See Also:**[TableRowSorter](http://docs.google.com/javax/swing/table/TableRowSorter.html), [DefaultTableModel](http://docs.google.com/javax/swing/table/DefaultTableModel.html), [Collator](http://docs.google.com/java/text/Collator.html)

| **Nested Class Summary** | |
| --- | --- |
| protected static class | [**DefaultRowSorter.ModelWrapper**](http://docs.google.com/javax/swing/DefaultRowSorter.ModelWrapper.html)**<**[**M**](http://docs.google.com/javax/swing/DefaultRowSorter.ModelWrapper.html)**,**[**I**](http://docs.google.com/javax/swing/DefaultRowSorter.ModelWrapper.html)**>**            DefaultRowSorter.ModelWrapper is responsible for providing the data that gets sorted by DefaultRowSorter. |

| **Nested classes/interfaces inherited from class javax.swing.**[**RowSorter**](http://docs.google.com/javax/swing/RowSorter.html) |
| --- |
| [RowSorter.SortKey](http://docs.google.com/javax/swing/RowSorter.SortKey.html) |

| **Constructor Summary** | |
| --- | --- |
| [**DefaultRowSorter**](http://docs.google.com/javax/swing/DefaultRowSorter.html#DefaultRowSorter())()            Creates an empty DefaultRowSorter. |

| **Method Summary** | |
| --- | --- |
| void | [**allRowsChanged**](http://docs.google.com/javax/swing/DefaultRowSorter.html#allRowsChanged())()            Invoked when the contents of the underlying model have completely changed. |
| int | [**convertRowIndexToModel**](http://docs.google.com/javax/swing/DefaultRowSorter.html#convertRowIndexToModel(int))(int index)            Returns the location of index in terms of the underlying model. |
| int | [**convertRowIndexToView**](http://docs.google.com/javax/swing/DefaultRowSorter.html#convertRowIndexToView(int))(int index)            Returns the location of index in terms of the view. |
| [Comparator](http://docs.google.com/java/util/Comparator.html)<?> | [**getComparator**](http://docs.google.com/javax/swing/DefaultRowSorter.html#getComparator(int))(int column)            Returns the Comparator for the specified column. |
| int | [**getMaxSortKeys**](http://docs.google.com/javax/swing/DefaultRowSorter.html#getMaxSortKeys())()            Returns the maximum number of sort keys. |
| [M](http://docs.google.com/javax/swing/DefaultRowSorter.html) | [**getModel**](http://docs.google.com/javax/swing/DefaultRowSorter.html#getModel())()            Returns the underlying model. |
| int | [**getModelRowCount**](http://docs.google.com/javax/swing/DefaultRowSorter.html#getModelRowCount())()            Returns the number of rows in the underlying model. |
| protected  [DefaultRowSorter.ModelWrapper](http://docs.google.com/javax/swing/DefaultRowSorter.ModelWrapper.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html),[I](http://docs.google.com/javax/swing/DefaultRowSorter.html)> | [**getModelWrapper**](http://docs.google.com/javax/swing/DefaultRowSorter.html#getModelWrapper())()            Returns the model wrapper providing the data that is being sorted and filtered. |
| [RowFilter](http://docs.google.com/javax/swing/RowFilter.html)<? super [M](http://docs.google.com/javax/swing/DefaultRowSorter.html),? super [I](http://docs.google.com/javax/swing/DefaultRowSorter.html)> | [**getRowFilter**](http://docs.google.com/javax/swing/DefaultRowSorter.html#getRowFilter())()            Returns the filter that determines which rows, if any, should be hidden from view. |
| [List](http://docs.google.com/java/util/List.html)<? extends [RowSorter.SortKey](http://docs.google.com/javax/swing/RowSorter.SortKey.html)> | [**getSortKeys**](http://docs.google.com/javax/swing/DefaultRowSorter.html#getSortKeys())()            Returns the current sort keys. |
| boolean | [**getSortsOnUpdates**](http://docs.google.com/javax/swing/DefaultRowSorter.html#getSortsOnUpdates())()            Returns true if a sort should happen when the underlying model is updated; otherwise, returns false. |
| int | [**getViewRowCount**](http://docs.google.com/javax/swing/DefaultRowSorter.html#getViewRowCount())()            Returns the number of rows in the view. |
| boolean | [**isSortable**](http://docs.google.com/javax/swing/DefaultRowSorter.html#isSortable(int))(int column)            Returns true if the specified column is sortable; otherwise, false. |
| void | [**modelStructureChanged**](http://docs.google.com/javax/swing/DefaultRowSorter.html#modelStructureChanged())()            Invoked when the underlying model structure has completely changed. |
| void | [**rowsDeleted**](http://docs.google.com/javax/swing/DefaultRowSorter.html#rowsDeleted(int,%20int))(int firstRow, int endRow)            Invoked when rows have been deleted from the underlying model in the specified range (inclusive). |
| void | [**rowsInserted**](http://docs.google.com/javax/swing/DefaultRowSorter.html#rowsInserted(int,%20int))(int firstRow, int endRow)            Invoked when rows have been inserted into the underlying model in the specified range (inclusive). |
| void | [**rowsUpdated**](http://docs.google.com/javax/swing/DefaultRowSorter.html#rowsUpdated(int,%20int))(int firstRow, int endRow)            Invoked when rows have been changed in the underlying model between the specified range (inclusive). |
| void | [**rowsUpdated**](http://docs.google.com/javax/swing/DefaultRowSorter.html#rowsUpdated(int,%20int,%20int))(int firstRow, int endRow, int column)            Invoked when the column in the rows have been updated in the underlying model between the specified range. |
| void | [**setComparator**](http://docs.google.com/javax/swing/DefaultRowSorter.html#setComparator(int,%20java.util.Comparator))(int column, [Comparator](http://docs.google.com/java/util/Comparator.html)<?> comparator)            Sets the Comparator to use when sorting the specified column. |
| void | [**setMaxSortKeys**](http://docs.google.com/javax/swing/DefaultRowSorter.html#setMaxSortKeys(int))(int max)            Sets the maximum number of sort keys. |
| protected  void | [**setModelWrapper**](http://docs.google.com/javax/swing/DefaultRowSorter.html#setModelWrapper(javax.swing.DefaultRowSorter.ModelWrapper))([DefaultRowSorter.ModelWrapper](http://docs.google.com/javax/swing/DefaultRowSorter.ModelWrapper.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html),[I](http://docs.google.com/javax/swing/DefaultRowSorter.html)> modelWrapper)            Sets the model wrapper providing the data that is being sorted and filtered. |
| void | [**setRowFilter**](http://docs.google.com/javax/swing/DefaultRowSorter.html#setRowFilter(javax.swing.RowFilter))([RowFilter](http://docs.google.com/javax/swing/RowFilter.html)<? super [M](http://docs.google.com/javax/swing/DefaultRowSorter.html),? super [I](http://docs.google.com/javax/swing/DefaultRowSorter.html)> filter)            Sets the filter that determines which rows, if any, should be hidden from the view. |
| void | [**setSortable**](http://docs.google.com/javax/swing/DefaultRowSorter.html#setSortable(int,%20boolean))(int column, boolean sortable)            Sets whether or not the specified column is sortable. |
| void | [**setSortKeys**](http://docs.google.com/javax/swing/DefaultRowSorter.html#setSortKeys(java.util.List))([List](http://docs.google.com/java/util/List.html)<? extends [RowSorter.SortKey](http://docs.google.com/javax/swing/RowSorter.SortKey.html)> sortKeys)            Sets the sort keys. |
| void | [**setSortsOnUpdates**](http://docs.google.com/javax/swing/DefaultRowSorter.html#setSortsOnUpdates(boolean))(boolean sortsOnUpdates)            If true, specifies that a sort should happen when the underlying model is updated (rowsUpdated is invoked). |
| void | [**sort**](http://docs.google.com/javax/swing/DefaultRowSorter.html#sort())()            Sorts and filters the rows in the view based on the sort keys of the columns currently being sorted and the filter, if any, associated with this sorter. |
| void | [**toggleSortOrder**](http://docs.google.com/javax/swing/DefaultRowSorter.html#toggleSortOrder(int))(int column)            Reverses the sort order from ascending to descending (or descending to ascending) if the specified column is already the primary sorted column; otherwise, makes the specified column the primary sorted column, with an ascending sort order. |
| protected  boolean | [**useToString**](http://docs.google.com/javax/swing/DefaultRowSorter.html#useToString(int))(int column)            Returns whether or not to convert the value to a string before doing comparisons when sorting. |

| **Methods inherited from class javax.swing.**[**RowSorter**](http://docs.google.com/javax/swing/RowSorter.html) |
| --- |
| [addRowSorterListener](http://docs.google.com/javax/swing/RowSorter.html#addRowSorterListener(javax.swing.event.RowSorterListener)), [fireRowSorterChanged](http://docs.google.com/javax/swing/RowSorter.html#fireRowSorterChanged(int%5B%5D)), [fireSortOrderChanged](http://docs.google.com/javax/swing/RowSorter.html#fireSortOrderChanged()), [removeRowSorterListener](http://docs.google.com/javax/swing/RowSorter.html#removeRowSorterListener(javax.swing.event.RowSorterListener)) |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [toString](http://docs.google.com/java/lang/Object.html#toString()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Constructor Detail** |
| --- |

### DefaultRowSorter

public **DefaultRowSorter**()

Creates an empty DefaultRowSorter.

| **Method Detail** |
| --- |

### setModelWrapper

protected final void **setModelWrapper**([DefaultRowSorter.ModelWrapper](http://docs.google.com/javax/swing/DefaultRowSorter.ModelWrapper.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html),[I](http://docs.google.com/javax/swing/DefaultRowSorter.html)> modelWrapper)

Sets the model wrapper providing the data that is being sorted and filtered.

**Parameters:**modelWrapper - the model wrapper responsible for providing the data that gets sorted and filtered **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if modelWrapper is null

### getModelWrapper

protected final [DefaultRowSorter.ModelWrapper](http://docs.google.com/javax/swing/DefaultRowSorter.ModelWrapper.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html),[I](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **getModelWrapper**()

Returns the model wrapper providing the data that is being sorted and filtered.

**Returns:**the model wrapper responsible for providing the data that gets sorted and filtered

### getModel

public final [M](http://docs.google.com/javax/swing/DefaultRowSorter.html) **getModel**()

Returns the underlying model.

**Specified by:**[getModel](http://docs.google.com/javax/swing/RowSorter.html#getModel()) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Returns:**the underlying model

### setSortable

public void **setSortable**(int column,  
 boolean sortable)

Sets whether or not the specified column is sortable. The specified value is only checked when toggleSortOrder is invoked. It is still possible to sort on a column that has been marked as unsortable by directly setting the sort keys. The default is true.

**Parameters:**column - the column to enable or disable sorting on, in terms of the underlying modelsortable - whether or not the specified column is sortable **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if column is outside the range of the model**See Also:**[toggleSortOrder(int)](http://docs.google.com/javax/swing/DefaultRowSorter.html#toggleSortOrder(int)), [setSortKeys(java.util.List)](http://docs.google.com/javax/swing/DefaultRowSorter.html#setSortKeys(java.util.List))

### isSortable

public boolean **isSortable**(int column)

Returns true if the specified column is sortable; otherwise, false.

**Parameters:**column - the column to check sorting for, in terms of the underlying model **Returns:**true if the column is sortable **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if column is outside the range of the underlying model

### setSortKeys

public void **setSortKeys**([List](http://docs.google.com/java/util/List.html)<? extends [RowSorter.SortKey](http://docs.google.com/javax/swing/RowSorter.SortKey.html)> sortKeys)

Sets the sort keys. This creates a copy of the supplied List; subsequent changes to the supplied List do not effect this DefaultRowSorter. If the sort keys have changed this triggers a sort.

**Specified by:**[setSortKeys](http://docs.google.com/javax/swing/RowSorter.html#setSortKeys(java.util.List)) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Parameters:**sortKeys - the new SortKeys; null is a shorthand for specifying an empty list, indicating that the view should be unsorted **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if any of the values in sortKeys are null or have a column index outside the range of the model

### getSortKeys

public [List](http://docs.google.com/java/util/List.html)<? extends [RowSorter.SortKey](http://docs.google.com/javax/swing/RowSorter.SortKey.html)> **getSortKeys**()

Returns the current sort keys. This returns an unmodifiable non-null List. If you need to change the sort keys, make a copy of the returned List, mutate the copy and invoke setSortKeys with the new list.

**Specified by:**[getSortKeys](http://docs.google.com/javax/swing/RowSorter.html#getSortKeys()) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Returns:**the current sort order

### setMaxSortKeys

public void **setMaxSortKeys**(int max)

Sets the maximum number of sort keys. The number of sort keys determines how equal values are resolved when sorting. For example, assume a table row sorter is created and setMaxSortKeys(2) is invoked on it. The user clicks the header for column 1, causing the table rows to be sorted based on the items in column 1. Next, the user clicks the header for column 2, causing the table to be sorted based on the items in column 2; if any items in column 2 are equal, then those particular rows are ordered based on the items in column 1. In this case, we say that the rows are primarily sorted on column 2, and secondarily on column 1. If the user then clicks the header for column 3, then the items are primarily sorted on column 3 and secondarily sorted on column 2. Because the maximum number of sort keys has been set to 2 with setMaxSortKeys, column 1 no longer has an effect on the order.

The maximum number of sort keys is enforced by toggleSortOrder. You can specify more sort keys by invoking setSortKeys directly and they will all be honored. However if toggleSortOrder is subsequently invoked the maximum number of sort keys will be enforced. The default value is 3.

**Parameters:**max - the maximum number of sort keys **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if max < 1

### getMaxSortKeys

public int **getMaxSortKeys**()

Returns the maximum number of sort keys.

**Returns:**the maximum number of sort keys

### setSortsOnUpdates

public void **setSortsOnUpdates**(boolean sortsOnUpdates)

If true, specifies that a sort should happen when the underlying model is updated (rowsUpdated is invoked). For example, if this is true and the user edits an entry the location of that item in the view may change. The default is false.

**Parameters:**sortsOnUpdates - whether or not to sort on update events

### getSortsOnUpdates

public boolean **getSortsOnUpdates**()

Returns true if a sort should happen when the underlying model is updated; otherwise, returns false.

**Returns:**whether or not to sort when the model is updated

### setRowFilter

public void **setRowFilter**([RowFilter](http://docs.google.com/javax/swing/RowFilter.html)<? super [M](http://docs.google.com/javax/swing/DefaultRowSorter.html),? super [I](http://docs.google.com/javax/swing/DefaultRowSorter.html)> filter)

Sets the filter that determines which rows, if any, should be hidden from the view. The filter is applied before sorting. A value of null indicates all values from the model should be included.

RowFilter's include method is passed an Entry that wraps the underlying model. The number of columns in the Entry corresponds to the number of columns in the ModelWrapper. The identifier comes from the ModelWrapper as well.

This method triggers a sort.

**Parameters:**filter - the filter used to determine what entries should be included

### getRowFilter

public [RowFilter](http://docs.google.com/javax/swing/RowFilter.html)<? super [M](http://docs.google.com/javax/swing/DefaultRowSorter.html),? super [I](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **getRowFilter**()

Returns the filter that determines which rows, if any, should be hidden from view.

**Returns:**the filter

### toggleSortOrder

public void **toggleSortOrder**(int column)

Reverses the sort order from ascending to descending (or descending to ascending) if the specified column is already the primary sorted column; otherwise, makes the specified column the primary sorted column, with an ascending sort order. If the specified column is not sortable, this method has no effect.

**Specified by:**[toggleSortOrder](http://docs.google.com/javax/swing/RowSorter.html#toggleSortOrder(int)) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Parameters:**column - index of the column to make the primary sorted column, in terms of the underlying model **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if column is outside the range of the underlying model**See Also:**[setSortable(int,boolean)](http://docs.google.com/javax/swing/DefaultRowSorter.html#setSortable(int,%20boolean)), [setMaxSortKeys(int)](http://docs.google.com/javax/swing/DefaultRowSorter.html#setMaxSortKeys(int))

### convertRowIndexToView

public int **convertRowIndexToView**(int index)

Returns the location of index in terms of the view. That is, for the row index in the coordinates of the underlying model this returns the row index in terms of the view.

**Specified by:**[convertRowIndexToView](http://docs.google.com/javax/swing/RowSorter.html#convertRowIndexToView(int)) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Parameters:**index - the row index in terms of the underlying model **Returns:**row index in terms of the view, or -1 if index has been filtered out of the view **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if index is outside the range of the model

### convertRowIndexToModel

public int **convertRowIndexToModel**(int index)

Returns the location of index in terms of the underlying model. That is, for the row index in the coordinates of the view this returns the row index in terms of the underlying model.

**Specified by:**[convertRowIndexToModel](http://docs.google.com/javax/swing/RowSorter.html#convertRowIndexToModel(int)) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Parameters:**index - the row index in terms of the underlying view **Returns:**row index in terms of the view **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if index is outside the range of the view

### sort

public void **sort**()

Sorts and filters the rows in the view based on the sort keys of the columns currently being sorted and the filter, if any, associated with this sorter. An empty sortKeys list indicates that the view should unsorted, the same as the model.

**See Also:**[setRowFilter(javax.swing.RowFilter)](http://docs.google.com/javax/swing/DefaultRowSorter.html#setRowFilter(javax.swing.RowFilter)), [setSortKeys(java.util.List)](http://docs.google.com/javax/swing/DefaultRowSorter.html#setSortKeys(java.util.List))

### useToString

protected boolean **useToString**(int column)

Returns whether or not to convert the value to a string before doing comparisons when sorting. If true ModelWrapper.getStringValueAt will be used, otherwise ModelWrapper.getValueAt will be used. It is up to subclasses, such as TableRowSorter, to honor this value in their ModelWrapper implementation.

**Parameters:**column - the index of the column to test, in terms of the underlying model **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if column is not valid

### setComparator

public void **setComparator**(int column,  
 [Comparator](http://docs.google.com/java/util/Comparator.html)<?> comparator)

Sets the Comparator to use when sorting the specified column. This does not trigger a sort. If you want to sort after setting the comparator you need to explicitly invoke sort.

**Parameters:**column - the index of the column the Comparator is to be used for, in terms of the underlying modelcomparator - the Comparator to use **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if column is outside the range of the underlying model

### getComparator

public [Comparator](http://docs.google.com/java/util/Comparator.html)<?> **getComparator**(int column)

Returns the Comparator for the specified column. This will return null if a Comparator has not been specified for the column.

**Parameters:**column - the column to fetch the Comparator for, in terms of the underlying model **Returns:**the Comparator for the specified column **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if column is outside the range of the underlying model

### getViewRowCount

public int **getViewRowCount**()

Returns the number of rows in the view. If the contents have been filtered this might differ from the row count of the underlying model.

**Specified by:**[getViewRowCount](http://docs.google.com/javax/swing/RowSorter.html#getViewRowCount()) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Returns:**number of rows in the view**See Also:**[RowSorter.getModelRowCount()](http://docs.google.com/javax/swing/RowSorter.html#getModelRowCount())

### getModelRowCount

public int **getModelRowCount**()

Returns the number of rows in the underlying model.

**Specified by:**[getModelRowCount](http://docs.google.com/javax/swing/RowSorter.html#getModelRowCount()) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Returns:**number of rows in the underlying model**See Also:**[RowSorter.getViewRowCount()](http://docs.google.com/javax/swing/RowSorter.html#getViewRowCount())

### modelStructureChanged

public void **modelStructureChanged**()

Invoked when the underlying model structure has completely changed. For example, if the number of columns in a TableModel changed, this method would be invoked.

You normally do not call this method. This method is public to allow view classes to call it.

**Specified by:**[modelStructureChanged](http://docs.google.com/javax/swing/RowSorter.html#modelStructureChanged()) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)>

### allRowsChanged

public void **allRowsChanged**()

Invoked when the contents of the underlying model have completely changed. The structure of the table is the same, only the contents have changed. This is typically sent when it is too expensive to characterize the change in terms of the other methods.

You normally do not call this method. This method is public to allow view classes to call it.

**Specified by:**[allRowsChanged](http://docs.google.com/javax/swing/RowSorter.html#allRowsChanged()) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)>

### rowsInserted

public void **rowsInserted**(int firstRow,  
 int endRow)

Invoked when rows have been inserted into the underlying model in the specified range (inclusive).

The arguments give the indices of the effected range. The first argument is in terms of the model before the change, and must be less than or equal to the size of the model before the change. The second argument is in terms of the model after the change and must be less than the size of the model after the change. For example, if you have a 5-row model and add 3 items to the end of the model the indices are 5, 7.

You normally do not call this method. This method is public to allow view classes to call it.

**Specified by:**[rowsInserted](http://docs.google.com/javax/swing/RowSorter.html#rowsInserted(int,%20int)) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Parameters:**firstRow - the first rowendRow - the last row **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if either argument is invalid, or firstRow > endRow

### rowsDeleted

public void **rowsDeleted**(int firstRow,  
 int endRow)

Invoked when rows have been deleted from the underlying model in the specified range (inclusive).

The arguments give the indices of the effected range and are in terms of the model **before** the change. For example, if you have a 5-row model and delete 3 items from the end of the model the indices are 2, 4.

You normally do not call this method. This method is public to allow view classes to call it.

**Specified by:**[rowsDeleted](http://docs.google.com/javax/swing/RowSorter.html#rowsDeleted(int,%20int)) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Parameters:**firstRow - the first rowendRow - the last row **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if either argument is outside the range of the model before the change, or firstRow > endRow

### rowsUpdated

public void **rowsUpdated**(int firstRow,  
 int endRow)

Invoked when rows have been changed in the underlying model between the specified range (inclusive).

You normally do not call this method. This method is public to allow view classes to call it.

**Specified by:**[rowsUpdated](http://docs.google.com/javax/swing/RowSorter.html#rowsUpdated(int,%20int)) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Parameters:**firstRow - the first row, in terms of the underlying modelendRow - the last row, in terms of the underlying model **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if either argument is outside the range of the underlying model, or firstRow > endRow

### rowsUpdated

public void **rowsUpdated**(int firstRow,  
 int endRow,  
 int column)

Invoked when the column in the rows have been updated in the underlying model between the specified range.

You normally do not call this method. This method is public to allow view classes to call it.

**Specified by:**[rowsUpdated](http://docs.google.com/javax/swing/RowSorter.html#rowsUpdated(int,%20int,%20int)) in class [RowSorter](http://docs.google.com/javax/swing/RowSorter.html)<[M](http://docs.google.com/javax/swing/DefaultRowSorter.html)> **Parameters:**firstRow - the first row, in terms of the underlying modelendRow - the last row, in terms of the underlying modelcolumn - the column that has changed, in terms of the underlying model **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if either argument is outside the range of the underlying model after the change, firstRow > endRow, or column is outside the range of the underlying model

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/DefaultRowSorter.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/swing/DefaultListSelectionModel.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/DefaultRowSorter.ModelWrapper.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/DefaultRowSorter.html)    [**NO FRAMES**](http://docs.google.com/DefaultRowSorter.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#3znysh7) | FIELD | [CONSTR](#tyjcwt) | [METHOD](#3dy6vkm) | DETAIL: FIELD | [CONSTR](#2s8eyo1) | [METHOD](#3rdcrjn) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).