Java™ Platform
Standard Ed. 8

All Classes All P

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfe
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi
java.awt.image

All Classes

AbstractAction
AbstractAnnotation\
AbstractAnnotation\
AbstractAnnotation\
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChoos
AbstractDocument
AbstractDocument.A
AbstractDocument.C
AbstractDocument.E
AbstractElementVisi
AbstractElementVisi
AbstractElementVisi
AbstractExecutorSe
AbstractInterruptibl
AbstractLayoutCach
AbstractLayoutCach
AbstractList
AbstractListModel
AbstractMap
AbstractMap.Simple
AbstractMap.Simple
AbstractMarshallerIı
AbstractMethodErro
AbstractOwnableSyı
AbstractPreferences
AbstractProcessor
AbstractQueue
AbstractQueuedLon
AbstractQueuedSyn
AbstractRegionPaint
AbstractRegionPaint
AbstractRegionPaint
AbstractScriptEngin
AbstractSelectableC

compact1, compact2, compact3

java.util

Class Vector<E>

java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractList<E>
            java.util.Vector<E>

All Implemented Interfaces:

`Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess`

Direct Known Subclasses:

`Stack`

```
public class Vector<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

The `Vector` class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a `Vector` can grow or shrink as needed to accommodate adding and removing items after the `Vector` has been created.

Each vector tries to optimize storage management by maintaining a `capacity` and a `capacityIncrement`. The `capacity` is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of `capacityIncrement`. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

The iterators returned by this class's `iterator` and `listIterator` methods are *fail-fast*: if the vector is structurally modified at any time after the iterator is created, in any way except through the iterator's own `remove` or `add` methods, the iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future. The `Enumerations` returned by the `elements` method are *not* fail-fast.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw `ConcurrentModificationException` on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*