---

package java.util

Interface Index

- Enumeration
- Observer

Class Index

- BitSet
- Date
- Dictionary
- Hashtable
- Observable
- Properties
- Random
- Stack
- StringTokenizer
- Vector

Exception Index

- EmptyStackException
- NoSuchElementException

Class java.util.BitSet

```
java.lang.Object
    |
    +----java.util.BitSet
```

---

public final class **BitSet**
extends Object
implements Cloneable

A set of bits. The set automatically grows as more bits are needed.

Constructor Index

- **BitSet**()
  Creates an empty set.
- **BitSet**(int)
  Creates an empty set with the specified size.

Method Index

- **and**(BitSet)
  Logically ANDs this bit set with the specified set of bits.
- **clear**(int)
  Clears a bit.
- **clone**()
  Clones the BitSet.
- **equals**(Object)
  Compares this object against the specified object.
- **get**(int)
  Gets a bit.
- **hashCode**()
  Gets the hashcode.
- **or**(BitSet)
  Logically ORs this bit set with the specified set of bits.
- **set**(int)
  Sets a bit.
- **size**()

Calculates and returns the set's size

- **toString**()
  Converts the BitSet to a String.
- **xor**(BitSet)
  Logically XORs this bit set with the specified set of bits.

Constructors

BitSet

```
public BitSet()
```

Creates an empty set.

BitSet

```
public BitSet(int nbits)
```

Creates an empty set with the specified size.
Parameters:
nbits – the size of the set

Methods

set

```
public void set(int bit)
```

Sets a bit.
Parameters:
bit – the bit to be set

clear

```
public void clear(int bit)
```

Clears a bit.
Parameters:
bit – the bit to be cleared

get

```
public boolean get(int bit)
```

Gets a bit.

Parameters:
bit – the bit to be gotten

and

```
public void and(BitSet set)
```

Logically ANDs this bit set with the specified set of bits.
Parameters:
set – the bit set to be ANDed with

or

```
public void or(BitSet set)
```

Logically ORs this bit set with the specified set of bits.
Parameters:
set – the bit set to be ORed with

xor

```
public void xor(BitSet set)
```

Logically XORs this bit set with the specified set of bits.
Parameters:
set – the bit set to be XORed with

hashCode

```
public int hashCode()
```

Gets the hashcode.
Overrides:
hashCode in class Object

size

```
public int size()
```

Calculates and returns the set's size

equals

```
public boolean equals(Object obj)
```

Compares this object against the specified object.
Parameters:
obj – the object to commpare with
Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object

clone

```
public Object clone()
```

Clones the BitSet.

Overrides:

clone in class Object

toString

```
public String toString()
```

Converts the BitSet to a String.

Overrides:

toString in class Object

Class java.util.Date

```
java.lang.Object
    |
    +----java.util.Date
```

---

public class **Date**
extends Object

A wrapper for a date. This class lets you manipulate dates in a system independent way. To print today's date use:

```
Date d = new Date();
System.out.println("today = " + d);
```

To find out what day corresponds to a particular date:

```
Date d = new Date(63, 0, 16);   // January 16, 1963
System.out.println("Day of the week: " + d.getDay());
```

The date can be set and examined according to the local time zone into the year, month, day, hour, minute and second.

While the API is intended to reflect UTC, Coordinated Universal Time, it doesn't do so exactly. This inexact behavior is inherited from the time system of the underlying OS. All modern OS's that I (jag) am aware of assume that 1 day = 24*60*60 seconds. In UTC, about once a year there is an extra second, called a "leap second" added to a day to account for the wobble of the earth. Most computer clocks are not accurate enough to be able to reflect this distinction. Some computer standards are defined in GMT, which is equivalent to UT, Universal Time. GMT is the "civil" name for the standard, UT is the "scientific" name for the same standard. The distinction between UTC and UT is that the first is based on an atomic clock and the second is based on astronomical observations, which for all practical purposes is an invisibly fine hair to split. An interesting source of further information is the US Naval Observatory, particularly the Directorate of Time and their definitions of Systems of Time.

Constructor Index

- **Date**()
  Creates today's date/time.
- **Date**(long)
  Creates a date.

- **Date**(int, int, int)
  Creates a date.
- **Date**(int, int, int, int, int)
  Creates a date.
- **Date**(int, int, int, int, int, int)
  Creates a date.
- **Date**(String)
  Creates a date from a string according to the syntax accepted by parse().

Method Index

- **UTC**(int, int, int, int, int, int)
  Calculates a UTC value from YMDHMS.
- **after**(Date)
  Checks whether this date comes after the specified date.
- **before**(Date)
  Checks whether this date comes before the specified date.
- **equals**(Object)
  Compares this object against the specified object.
- **getDate**()
  Returns the day of the month.
- **getDay**()
  Returns the day of the week.
- **getHours**()
  Returns the hour.
- **getMinutes**()
  Returns the minute.
- **getMonth**()
  Returns the month.
- **getSeconds**()
  Returns the second.
- **getTime**()
  Returns the time in milliseconds since the epoch.
- **getTimezoneOffset**()
  Return the time zone offset in minutes for the current locale that is appropriate for this time.
- **getYear**()
  Returns the year after 1900.
- **hashCode**()
  Computes a hashCode.
- **parse**(String)
  Given a string representing a time, parse it and return the time value.
- **setDate**(int)
  Sets the date.
- **setHours**(int)
  Sets the hours.
- **setMinutes**(int)
  Sets the minutes.

- **setMonth**(int)
  Sets the month.
- **setSeconds**(int)
  Sets the seconds.
- **setTime**(long)
  Sets the time.
- **setYear**(int)
  Sets the year.
- **toGMTString**()
  Converts a date to a String, using the Internet GMT conventions.
- **toLocaleString**()
  Converts a date to a String, using the locale conventions.
- **toString**()
  Converts a date to a String, using the UNIX ctime conventions.

Constructors

Date

```
public Date()
```

Creates today's date/time.

Date

```
public Date(long date)
```

Creates a date. The fields are normalized before the Date object is created. The argument does not have to be in the correct range. For example, the 32nd of January is correctly interpreted as the 1st of February. You can use this to figure out what day a particular date falls on.

Parameters:

date – the value of the argument to be created

Date

```
public Date(int year,
            int month,
            int date)
```

Creates a date. The fields are normalized before the Date object is created. The arguments do not have to be in the correct range. For example, the 32nd of January is correctly interpreted as the 1st of February. You can use this to figure out what day a particular date falls on.

Parameters:

year – a year after 1900
month – a month between 0–11
date – day of the month between 1–31

Date

```
public Date(int year,
            int month,
            int date,
            int hrs,
            int min)
```

Creates a date. The fields are normalized before the Date object is created. The arguments do not have to be in the correct range. For example, the 32nd of January is correctly interpreted as the 1st of February. You can use this to figure out what day a particular date falls on.

Parameters:

year – a year after 1900
month – a month between 0–11
date – day of the month between 1–31
hrs – hours between 0–23
min – minutes between 0–59

Date

```
public Date(int year,
            int month,
            int date,
            int hrs,
            int min,
            int sec)
```

Creates a date. The fields are normalized before the Date object is created. The arguments do not have to be in the correct range. For example, the 32nd of January is correctly interpreted as the 1st of February. You can use this to figure out what day a particular date falls on.

Parameters:

year – a year after 1900
month – a month between 0–11
date – day of the month between 1–31
hrs – hours between 0–23
min – minutes between 0–59
sec – seconds between 0–59

Date

```
public Date(String s)
```

Creates a date from a string according to the syntax accepted by parse().

Methods

UTC

```
public static long UTC(int year,
```

```
                    int month,
                    int date,
                    int hrs,
                    int min,
                    int sec)
```

Calculates a UTC value from YMDHMS. Interpretes the parameters in UTC, *not in the local time zone.*

Parameters:

year − a year after 1900
month − a month between 0−11
date − day of the month between 1−31
hrs − hours between 0−23
min − minutes between 0−59
sec − seconds between 0−59

parse

```
public static long parse(String s)
```

Given a string representing a time, parse it and return the time value. It accepts many syntaxes, but most importantly, in accepts the IETF standard date syntax: "Sat, 12 Aug 1995 13:30:00 GMT". It understands the continental US time zone abbreviations, but for general use, a timezone offset should be used: "Sat, 12 Aug 1995 13:30:00 GMT+0430" (4 hours, 30 minutes west of the Greenwich meridian). If no time zone is specified, the local time zone is assumed. GMT and UTC are considered equivalent.

getYear

```
public int getYear()
```

Returns the year after 1900.

setYear

```
public void setYear(int year)
```

Sets the year.

Parameters:

year − the year value

getMonth

```
public int getMonth()
```

Returns the month. This method assigns months with the values 0−11, with January beginning at value 0.

setMonth

```
public void setMonth(int month)
```

Sets the month.
Parameters:
month – the month value (0–11)

getDate

```
public int getDate()
```

Returns the day of the month. This method assigns days with the values of 1 to 31.

setDate

```
public void setDate(int date)
```

Sets the date.
Parameters:
date – the day value

getDay

```
public int getDay()
```

Returns the day of the week. This method assigns days of the week with the values 0–6, with 0 being Sunday.

getHours

```
public int getHours()
```

Returns the hour. This method assigns the value of the hours of the day to range from 0 to 23, with midnight equal to 0.

setHours

```
public void setHours(int hours)
```

Sets the hours.
Parameters:
hours – the hour value

getMinutes

```
public int getMinutes()
```

Returns the minute. This method assigns the minutes of an hour to be any value from 0 to 59.

setMinutes

```
public void setMinutes(int minutes)
```

Sets the minutes.
Parameters:
minutes – the value of the minutes

getSeconds

```
public int getSeconds()
```

Returns the second. This method assigns the seconds of a minute to values of 0–59.

setSeconds

```
public void setSeconds(int seconds)
```

Sets the seconds.
Parameters:
seconds – the second value

getTime

```
public long getTime()
```

Returns the time in milliseconds since the epoch.

setTime

```
public void setTime(long time)
```

Sets the time.
Parameters:
time – The new time value in milliseconds since the epoch.

before

```
public boolean before(Date when)
```

Checks whether this date comes before the specified date.
Parameters:
when – the date to compare
Returns:
true if the original date comes before the specified one; false otherwise.

after

```
public boolean after(Date when)
```

Checks whether this date comes after the specified date.

Parameters:

when – the date to compare

Returns:

true if the original date comes after the specified one; false otherwise.

equals

```
public boolean equals(Object obj)
```

Compares this object against the specified object.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object

hashCode

```
public int hashCode()
```

Computes a hashCode.

Overrides:

hashCode in class Object

toString

```
public String toString()
```

Converts a date to a String, using the UNIX ctime conventions.

Overrides:

toString in class Object

toLocaleString

```
public String toLocaleString()
```

Converts a date to a String, using the locale conventions.

toGMTString

```
public String toGMTString()
```

Converts a date to a String, using the Internet GMT conventions.

getTimezoneOffset

```
public int getTimezoneOffset()
```

Return the time zone offset in minutes for the current locale that is appropriate for this time. This value would be a constant except for daylight savings time.

Class java.util.Dictionary

```
java.lang.Object
    |
    +----java.util.Dictionary
```

---

public class **Dictionary**
extends Object

The Dictionary class is the abstract parent of Hashtable, which maps keys to values. Any object can be used as a key and/or value.

See Also:
Hashtable, hashCode, equals

Constructor Index

- **Dictionary**()

Method Index

- **elements**()
  Returns an enumeration of the elements.
- **get**(Object)
  Gets the object associated with the specified key in the Dictionary.
- **isEmpty**()
  Returns true if the Dictionary contains no elements.
- **keys**()
  Returns an enumeration of the Dictionary's keys.
- **put**(Object, Object)
  Puts the specified element into the Dictionary, using the specified key.
- **remove**(Object)
  Removes the element corresponding to the key.
- **size**()
  Returns the number of elements contained within the Dictionary.

Constructors

Dictionary

```
public Dictionary()
```

Methods

size

```
public abstract int size()
```

Returns the number of elements contained within the Dictionary.

isEmpty

```
public abstract boolean isEmpty()
```

Returns true if the Dictionary contains no elements.

keys

```
public abstract Enumeration keys()
```

Returns an enumeration of the Dictionary's keys.
See Also:
elements, Enumeration

elements

```
public abstract Enumeration elements()
```

Returns an enumeration of the elements. Use the Enumeration methods on the returned object to fetch the elements sequentially.
See Also:
keys, Enumeration

get

```
public abstract Object get(Object key)
```

Gets the object associated with the specified key in the Dictionary.
Parameters:
key – the key in the hash table
Returns:
s the element for the key, or null if the key is not defined in the hash table.

See Also:

put

put

```
public abstract Object put(Object key,
                           Object value)
```

Puts the specified element into the Dictionary, using the specified key. The element may be retrieved by doing a get() with the same key. The key and the element cannot be null.

Parameters:

key – the specified hashtable key

value – the specified element

Returns:

the old value of the key, or null if it did not have one.

Throws: NullPointerException

If the value of the specified element is null.

See Also:

get

remove

```
public abstract Object remove(Object key)
```

Removes the element corresponding to the key. Does nothing if the key is not present.

Parameters:

key – the key that needs to be removed

Returns:

the value of key, or null if the key was not found.

Class java.util.EmptyStackException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.util.EmptyStackException
```

---

public class **EmptyStackException**
extends RuntimeException

Signals that the stack is empty.

See Also:
Stack

Constructor Index

- **EmptyStackException**()
  Constructs a new EmptyStackException with no detail message.

Constructors

EmptyStackException

```
public EmptyStackException()
```

Constructs a new EmptyStackException with no detail message. A detail message is a String that describes the exception.

Interface java.util.Enumeration

public interface **Enumeration**
extends Object

The Enumeration interface specifies a set of methods that may be used to enumerate, or count through, a set of values. The enumeration is consumed by use; its values may only be counted once.

For example, to print all elements of a Vector v:

```
for (Enumeration e = v.elements() ; e.hasMoreElements() ;) {
    System.out.println(e.nextElement());
}
```

See Also:
Vector, Hashtable

Method Index

- **hasMoreElements**()
  Returns true if the enumeration contains more elements; false if its empty.
- **nextElement**()
  Returns the next element of the enumeration.

Methods

hasMoreElements

```
public abstract boolean hasMoreElements()
```

Returns true if the enumeration contains more elements; false if its empty.

nextElement

```
public abstract Object nextElement()
```

Returns the next element of the enumeration. Calls to this method will enumerate successive elements.
Throws: NoSuchElementException
If no more elements exist.

Class java.util.Hashtable

```
java.lang.Object
   |
   +----java.util.Dictionary
           |
           +----java.util.Hashtable
```

---

public class **Hashtable**
extends Dictionary
implements Cloneable

Hashtable class. Maps keys to values. Any object can be used as a key and/or value.

To sucessfully store and retrieve objects from a hash table, the object used as the key must implement the hashCode() and equals() methods.

This example creates a hashtable of numbers. It uses the names of the numbers as keys:

```
Hashtable numbers = new Hashtable();
numbers.put("one", new Integer(1));
numbers.put("two", new Integer(2));
numbers.put("three", new Integer(3));
```

To retrieve a number use:

```
Integer n = (Integer)numbers.get("two");
if (n != null) {
    System.out.println("two = " + n);
}
```

See Also:
hashCode, equals

Constructor Index

- **Hashtable**(int, float)
  Constructs a new, empty hashtable with the specified initial capacity and the specified load factor.
- **Hashtable**(int)
  Constructs a new, empty hashtable with the specified initial capacity.

- **Hashtable**()
  Constructs a new, empty hashtable.

Method Index

- **clear**()
  Clears the hash table so that it has no more elements in it.
- **clone**()
  Creates a clone of the hashtable.
- **contains**(Object)
  Returns true if the specified object is an element of the hashtable.
- **containsKey**(Object)
  Returns true if the collection contains an element for the key.
- **elements**()
  Returns an enumeration of the elements.
- **get**(Object)
  Gets the object associated with the specified key in the hashtable.
- **isEmpty**()
  Returns true if the hashtable contains no elements.
- **keys**()
  Returns an enumeration of the hashtable's keys.
- **put**(Object, Object)
  Puts the specified element into the hashtable, using the specified key.
- **rehash**()
  Rehashes the content of the table into a bigger table.
- **remove**(Object)
  Removes the element corresponding to the key.
- **size**()
  Returns the number of elements contained in the hashtable.
- **toString**()
  Converts to a rather lengthy String.

Constructors

Hashtable

```
public Hashtable(int initialCapacity,
                 float loadFactor)
```

Constructs a new, empty hashtable with the specified initial capacity and the specified load factor.

Parameters:

initialCapacity – the initial number of buckets
loadFactor – a number between 0.0 and 1.0, it defines the threshold for rehashing the hashtable into a bigger one.

Throws: IllegalArgumentException

If the initial capacity is less than or equal to zero.

Throws: IllegalArgumentException

If the load factor is less than or equal to zero.

Hashtable

```
public Hashtable(int initialCapacity)
```

Constructs a new, empty hashtable with the specified initial capacity.

Parameters:

initialCapacity – the initial number of buckets

Hashtable

```
public Hashtable()
```

Constructs a new, empty hashtable. A default capacity and load factor is used. Note that the hashtable will automatically grow when it gets full.

Methods

size

```
public int size()
```

Returns the number of elements contained in the hashtable.

Overrides:

size in class Dictionary

isEmpty

```
public boolean isEmpty()
```

Returns true if the hashtable contains no elements.

Overrides:

isEmpty in class Dictionary

keys

```
public synchronized Enumeration keys()
```

Returns an enumeration of the hashtable's keys.

Overrides:

keys in class Dictionary

See Also:

elements, Enumeration

elements

```
public synchronized Enumeration elements()
```

Returns an enumeration of the elements. Use the Enumeration methods on the returned object to fetch the elements sequentially.

Overrides:

elements in class Dictionary

See Also:

keys, Enumeration

contains

```
public synchronized boolean contains(Object value)
```

Returns true if the specified object is an element of the hashtable. This operation is more expensive than the containsKey() method.

Parameters:

value – the value that we are looking for

Throws: NullPointerException

If the value being searched for is equal to null.

See Also:

containsKey

containsKey

```
public synchronized boolean containsKey(Object key)
```

Returns true if the collection contains an element for the key.

Parameters:

key – the key that we are looking for

See Also:

contains

get

```
public synchronized Object get(Object key)
```

Gets the object associated with the specified key in the hashtable.

Parameters:

key – the specified key

Returns:

s the element for the key or null if the key is not defined in the hash table.

Overrides:

get in class Dictionary

See Also:

put

rehash

```
protected void rehash()
```

Rehashes the content of the table into a bigger table. This method is called automatically when the hashtable's size exceeds the threshold.

put

```
public synchronized Object put(Object key,
                               Object value)
```

Puts the specified element into the hashtable, using the specified key. The element may be retrieved by doing a get() with the same key. The key and the element cannot be null.

Parameters:

key – the specified key in the hashtable
value – the specified element

Returns:

the old value of the key, or null if it did not have one.

Throws: NullPointerException

If the value of the element is equal to null.

Overrides:

put in class Dictionary

See Also:

get

remove

```
public synchronized Object remove(Object key)
```

Removes the element corresponding to the key. Does nothing if the key is not present.

Parameters:

key – the key that needs to be removed

Returns:

the value of key, or null if the key was not found.

Overrides:

remove in class Dictionary

clear

```
public synchronized void clear()
```

Clears the hash table so that it has no more elements in it.

clone

```
public synchronized Object clone()
```

Creates a clone of the hashtable. A shallow copy is made, the keys and elements themselves are

NOT cloned. This is a relatively expensive operation.

Overrides:

clone in class Object

toString

```
public synchronized String toString()
```

Converts to a rather lengthy String.

Overrides:

toString in class Object

Class java.util.NoSuchElementException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.util.NoSuchElementException
```

---

public class **NoSuchElementException**
extends RuntimeException

Signals that an enumeration is empty.

See Also:
Enumeration

Constructor Index

- **NoSuchElementException**()
  Constructs a NoSuchElementException with no detail message.
- **NoSuchElementException**(String)
  Constructs a NoSuchElementException with the specified detail message.

Constructors

NoSuchElementException

```
public NoSuchElementException()
```

Constructs a NoSuchElementException with no detail message. A detail message is a String that describes this particular exception.

NoSuchElementException

```
public NoSuchElementException(String s)
```

Constructs a NoSuchElementException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:

s – the detail message

Class java.util.Observable

```
java.lang.Object
   |
   +----java.util.Observable
```

---

public class **Observable**
extends Object

This class should be subclassed by observable object, or "data" in the Model–View paradigm. An Observable object may have any number of Observers. Whenever the Observable instance changes, it notifies all of its observers. Notification is done by calling the update() method on all observers.

Constructor Index

- **Observable**()

Method Index

- **addObserver**(Observer)
  Adds an observer to the observer list.
- **clearChanged**()
  Clears an observable change.
- **countObservers**()
  Counts the number of observers.
- **deleteObserver**(Observer)
  Deletes an observer from the observer list.
- **deleteObservers**()
  Deletes observers from the observer list.
- **hasChanged**()
  Returns a true boolean if an observable change has occurred.
- **notifyObservers**()
  Notifies all observers if an observable change occurs.
- **notifyObservers**(Object)
  Notifies all observers of the specified observable change which occurred.
- **setChanged**()
  Sets a flag to note an observable change.

Constructors

Observable

```
public Observable()
```

Methods

addObserver

```
public synchronized void addObserver(Observer o)
```

Adds an observer to the observer list.
Parameters:
o – the observer to be added

deleteObserver

```
public synchronized void deleteObserver(Observer o)
```

Deletes an observer from the observer list.
Parameters:
o – the observer to be deleted

notifyObservers

```
public void notifyObservers()
```

Notifies all observers if an observable change occurs.

notifyObservers

```
public synchronized void notifyObservers(Object arg)
```

Notifies all observers of the specified observable change which occurred.
Parameters:
arg – what is being notified

deleteObservers

```
public synchronized void deleteObservers()
```

Deletes observers from the observer list.

setChanged

```
protected synchronized void setChanged()
```

Sets a flag to note an observable change.

clearChanged

```
protected synchronized void clearChanged()
```

Clears an observable change.

hasChanged

```
public synchronized boolean hasChanged()
```

Returns a true boolean if an observable change has occurred.

countObservers

```
public synchronized int countObservers()
```

Counts the number of observers.

Interface java.util.Observer

public interface **Observer**
extends Object

When implemented, this interface allows all classes to be observable by instances of class Observer.

Method Index

- **update**(Observable, Object)
  Called when observers in the observable list need to be updated.

Methods

update

```
public abstract void update(Observable o,
                            Object arg)
```

Called when observers in the observable list need to be updated.
Parameters:
o – the list of observers
arg – the argument being notified

Class java.util.Properties

```
java.lang.Object
    |
    +----java.util.Dictionary
             |
             +----java.util.Hashtable
                      |
                      +----java.util.Properties
```

---

public class **Properties**
extends Hashtable

Persistent properties class. This class is basically a hashtable that can be saved/loaded from a stream. If a property is not found, a property list containing defaults is searched. This allows arbitrary nesting.

Variable Index

- **defaults**

Constructor Index

- **Properties**()
  Creates an empty property list.
- **Properties**(Properties)
  Creates an empty property list with specified defaults.

Method Index

- **getProperty**(String)
  Gets a property with the specified key.
- **getProperty**(String, String)
  Gets a property with the specified key and default.
- **list**(PrintStream)
  List properties, for debugging
- **load**(InputStream)
  Loads properties from an InputStream.

- **propertyNames**()
  Enumerates all the keys.
- **save**(OutputStream, String)
  Save properties to an OutputStream.

Variables

defaults

```
protected Properties defaults
```

Constructors

Properties

```
public Properties()
```

Creates an empty property list.

Properties

```
public Properties(Properties defaults)
```

Creates an empty property list with specified defaults.
Parameters:
defaults – the defaults

Methods

load

```
public synchronized void load(InputStream in) throws IOException
```

Loads properties from an InputStream.
Parameters:
in – the input stream
Throws:IOException
Error when reading from input stream.

save

```
public synchronized void save(OutputStream out,
                              String header)
```

Save properties to an OutputStream. Use the header as a comment at the top of the file.

getProperty

```
public String getProperty(String key)
```

Gets a property with the specified key. If the key is not found in this property list, tries the defaults. This method returns null if the property is not found.

Parameters:

key – the hashtable key

getProperty

```
public String getProperty(String key,
                          String defaultValue)
```

Gets a property with the specified key and default. If the key is not found in this property list, tries the defaults. This method returns defaultValue if the property is not found.

propertyNames

```
public Enumeration propertyNames()
```

Enumerates all the keys.

list

```
public void list(PrintStream out)
```

List properties, for debugging

Class java.util.Random

```
java.lang.Object
   |
   +----java.util.Random
```

---

public class **Random**
extends Object

A Random class generates a stream of pseudo−random numbers.

To create a new random number generator, use one of the following methods:

```
new Random()
new Random(long seed)
```

The form `new Random()` initializes the generator to a value based on the current time. The form `new Random(long seed)` seeds the random number generator with a specific initial value; use this if an application requires a repeatable stream of pseudo−random numbers.

The random number generator uses a 48−bit seed, which is modified using a linear congruential formula. See Donald Knuth, *The Art of Computer Programming, Volume 2*, Section 3.2.1. The generator's seed can be reset with the following method:

```
setSeed(long seed)
```

To create a pseudo−random number, use one of the following functions:

```
nextInt()
nextLong()
nextFloat()
nextDouble()
nextGaussian()
```

See Also:
random

Constructor Index

- **Random**()

Creates a new random number generator.

- **Random**(long)
  Creates a new random number generator using a single `long` seed.

Method Index

- **nextDouble**()
  Generates a pseudorandom uniformally distributed `double` value between 0.0 and 1.0.
- **nextFloat**()
  Generates a pseudorandom uniformally distributed `float` value between 0.0 and 1.0.
- **nextGaussian**()
  Generates a pseudorandom Gaussian distributed `double` value with mean 0.0 and standard deviation 1.0.
- **nextInt**()
  Generates a pseudorandom uniformally distributed `int` value.
- **nextLong**()
  Generate a pseudorandom uniformally distributed `long` value.
- **setSeed**(long)
  Sets the seed of the random number generator using a single `long` seed.

Constructors

Random

```
public Random()
```

Creates a new random number generator. Its seed will be initialized to a value based on the current time.

Random

```
public Random(long seed)
```

Creates a new random number generator using a single `long` seed.

Parameters:

seed – the initial seed

See Also:

setSeed

Methods

setSeed

```
public synchronized void setSeed(long seed)
```

Sets the seed of the random number generator using a single `long` seed.

Parameters:

seed – the initial seed

nextInt

```
public int nextInt()
```

Generates a pseudorandom uniformally distributed `int` value.

Returns:

an integer value.

nextLong

```
public long nextLong()
```

Generate a pseudorandom uniformally distributed `long` value.

Returns:

A long integer value

nextFloat

```
public float nextFloat()
```

Generates a pseudorandom uniformally distributed `float` value between 0.0 and 1.0.

Returns:

a `float` between 0.0 and 1.0 .

nextDouble

```
public double nextDouble()
```

Generates a pseudorandom uniformally distributed `double` value between 0.0 and 1.0.

Returns:

a `float` between 0.0 and 1.0 .

nextGaussian

```
public synchronized double nextGaussian()
```

Generates a pseudorandom Gaussian distributed `double` value with mean 0.0 and standard deviation 1.0.

Returns:

a Gaussian distributed `double`.

Class java.util.Stack

```
java.lang.Object
   |
   +----java.util.Vector
           |
           +----java.util.Stack
```

---

public class **Stack**
extends Vector

A Last-In-First-Out(LIFO) stack of objects.

Constructor Index

- **Stack**()

Method Index

- **empty**()
  Returns true if the stack is empty.
- **peek**()
  Peeks at the top of the stack.
- **pop**()
  Pops an item off the stack.
- **push**(Object)
  Pushes an item onto the stack.
- **search**(Object)
  Sees if an object is on the stack.

Constructors

Stack

```
public Stack()
```

Methods

push

```
public Object push(Object item)
```

Pushes an item onto the stack.
Parameters:
item – the item to be pushed on.

pop

```
public Object pop()
```

Pops an item off the stack.
Throws: EmptyStackException
If the stack is empty.

peek

```
public Object peek()
```

Peeks at the top of the stack.
Throws: EmptyStackException
If the stack is empty.

empty

```
public boolean empty()
```

Returns true if the stack is empty.

search

```
public int search(Object o)
```

Sees if an object is on the stack.
Parameters:
o – the desired object
Returns:
the distance from the top, or –1 if it is not found.

Class java.util.StringTokenizer

```
java.lang.Object
   |
   +----java.util.StringTokenizer
```

---

public class **StringTokenizer**
extends Object
implements Enumeration

StringTokenizer is a class that controls simple linear tokenization of a String. The set of delimiters, which defaults to common whitespace characters, may be specified at creation time or on a per−token basis.

Example usage:

```
String s = "this is a test";
StringTokenizer st = new StringTokenizer(s);
while (st.hasMoreTokens()) {
        println(st.nextToken());
}
```

Prints the following on the console:

```
this
is
a
test
```

Constructor Index

- **StringTokenizer**(String, String, boolean)
  Constructs a StringTokenizer on the specified String, using the specified delimiter set.
- **StringTokenizer**(String, String)
  Constructs a StringTokenizer on the specified String, using the specified delimiter set.
- **StringTokenizer**(String)
  Constructs a StringTokenizer on the specified String, using the default delimiter set (which is " \t\n\r").

Method Index

- **countTokens**()
  Returns the next number of tokens in the String using the current deliminter set.
- **hasMoreElements**()
  Returns true if the Enumeration has more elements.
- **hasMoreTokens**()
  Returns true if more tokens exist.
- **nextElement**()
  Returns the next element in the Enumeration.
- **nextToken**()
  Returns the next token of the String.
- **nextToken**(String)
  Returns the next token, after switching to the new delimiter set.

Constructors

StringTokenizer

```
public StringTokenizer(String str,
                       String delim,
                       boolean returnTokens)
```

Constructs a StringTokenizer on the specified String, using the specified delimiter set.
Parameters:
str – the input String
delim – the delimiter String
returnTokens – returns delimiters as tokens or skip them

StringTokenizer

```
public StringTokenizer(String str,
                       String delim)
```

Constructs a StringTokenizer on the specified String, using the specified delimiter set.
Parameters:
str – the input String
delim – the delimiter String

StringTokenizer

```
public StringTokenizer(String str)
```

Constructs a StringTokenizer on the specified String, using the default delimiter set (which is " \t\n\r").
Parameters:

str – the String

Methods

hasMoreTokens

```
public boolean hasMoreTokens()
```

Returns true if more tokens exist.

nextToken

```
public String nextToken()
```

Returns the next token of the String.
Throws: NoSuchElementException
If there are no more tokens in the String.

nextToken

```
public String nextToken(String delim)
```

Returns the next token, after switching to the new delimiter set. The new delimiter set remains the default after this call.
Parameters:
delim – the new delimiters

hasMoreElements

```
public boolean hasMoreElements()
```

Returns true if the Enumeration has more elements.

nextElement

```
public Object nextElement()
```

Returns the next element in the Enumeration.
Throws: NoSuchElementException
If there are no more elements in the enumeration.

countTokens

```
public int countTokens()
```

Returns the next number of tokens in the String using the current deliminter set. This is the number of times nextToken() can return before it will generate an exception. Use of this routine

to count the number of tokens is faster than repeatedly calling nextToken() because the substrings are not constructed and returned for each token.

Class java.util.Vector

```
java.lang.Object
    |
    +----java.util.Vector
```

---

public class **Vector**
extends Object
implements Cloneable

Vector class (a growable array).

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 elements are added to the vector, the vector's storage increases in chunks the size of capacityIncrement. Setting the capacity to what you want before inserting a large number of objects will reduce the amount of incremental reallocation. You can safely ignore the capacity and the vector will still work correctly.

Variable Index

- **capacityIncrement**
  The size of the increment.
- **elementCount**
  The number of elements in the buffer.
- **elementData**
  The buffer where elements are stored.

Constructor Index

- **Vector**(int, int)
  Constructs an empty vector with the specified storage capacity and the specified capacityIncrement.
- **Vector**(int)
  Constructs an empty vector with the specified storage capacity.
- **Vector**()
  Constructs an empty vector.

Method Index

- **addElement**(Object)
  Adds the specified object as the last element of the vector.
- **capacity**()
  Returns the current capacity of the vector.
- **clone**()
  Clones this vector.
- **contains**(Object)
  Returns true if the specified object is a value of the collection.
- **copyInto**(Object[])
  Copies the elements of this vector into the specified array.
- **elementAt**(int)
  Returns the element at the specified index.
- **elements**()
  Returns an enumeration of the elements.
- **ensureCapacity**(int)
  Ensures that the vector has at least the specified capacity.
- **firstElement**()
  Returns the first element of the sequence.
- **indexOf**(Object)
  Searches for the specified object, starting from the first position and returns an index to it.
- **indexOf**(Object, int)
  Searches for the specified object, starting at the specified position and returns an index to it.
- **insertElementAt**(Object, int)
  Inserts the specified object as an element at the specified index.
- **isEmpty**()
  Returns true if the collection contains no values.
- **lastElement**()
  Returns the last element of the sequence.
- **lastIndexOf**(Object)
  Searches backwards for the specified object, starting from the last position and returns an index to it.
- **lastIndexOf**(Object, int)
  Searches backwards for the specified object, starting from the specified position and returns an index to it.
- **removeAllElements**()
  Removes all elements of the vector.
- **removeElement**(Object)
  Removes the element from the vector.
- **removeElementAt**(int)
  Deletes the element at the specified index.
- **setElementAt**(Object, int)
  Sets the element at the specified index to be the specified object.
- **setSize**(int)
  Sets the size of the vector.
- **size**()
  Returns the number of elements in the vector.

- **toString**()
  Converts the vector to a string.
- **trimToSize**()
  Trims the vector's capacity down to size.

Variables

elementData

```
protected Object elementData[]
```

The buffer where elements are stored.

elementCount

```
protected int elementCount
```

The number of elements in the buffer.

capacityIncrement

```
protected int capacityIncrement
```

The size of the increment. If it is 0 the size of the the buffer is doubled everytime it needs to grow.

Constructors

Vector

```
public Vector(int initialCapacity,
              int capacityIncrement)
```

Constructs an empty vector with the specified storage capacity and the specified capacityIncrement.
Parameters:
initialCapacity – the initial storage capacity of the vector
capacityIncrement – how much to increase the element's size by.

Vector

```
public Vector(int initialCapacity)
```

Constructs an empty vector with the specified storage capacity.
Parameters:
initialCapacity – the initial storage capacity of the vector

Vector

```
public Vector()
```

Constructs an empty vector.

Methods

copyInto

```
public final synchronized void copyInto(Object anArray[])
```

Copies the elements of this vector into the specified array.
Parameters:
anArray – the array where elements get copied into

trimToSize

```
public final synchronized void trimToSize()
```

Trims the vector's capacity down to size. Use this operation to minimize the storage of a vector. Subsequent insertions will cause reallocation.

ensureCapacity

```
public final synchronized void ensureCapacity(int minCapacity)
```

Ensures that the vector has at least the specified capacity.
Parameters:
minCapacity – the desired minimum capacity

setSize

```
public final synchronized void setSize(int newSize)
```

Sets the size of the vector. If the size shrinks, the extra elements (at the end of the vector) are lost; if the size increases, the new elements are set to null.
Parameters:
newSize – the new size of the vector

capacity

```
public final int capacity()
```

Returns the current capacity of the vector.

size

```
public final int size()
```

Returns the number of elements in the vector. Note that this is not the same as the vector's capacity.

isEmpty

```
public final boolean isEmpty()
```

Returns true if the collection contains no values.

elements

```
public final synchronized Enumeration elements()
```

Returns an enumeration of the elements. Use the Enumeration methods on the returned object to fetch the elements sequentially.

contains

```
public final boolean contains(Object elem)
```

Returns true if the specified object is a value of the collection.

Parameters:

elem – the desired element

indexOf

```
public final int indexOf(Object elem)
```

Searches for the specified object, starting from the first position and returns an index to it.

Parameters:

elem – the desired element

Returns:

the index of the element, or –1 if it was not found.

indexOf

```
public final synchronized int indexOf(Object elem,
                                       int index)
```

Searches for the specified object, starting at the specified position and returns an index to it.

Parameters:

elem – the desired element

index – the index where to start searching

Returns:

the index of the element, or –1 if it was not found.

lastIndexOf

```
public final int lastIndexOf(Object elem)
```

Searches backwards for the specified object, starting from the last position and returns an index to it.

Parameters:

elem – the desired element

Returns:

the index of the element, or –1 if it was not found.

lastIndexOf

```
public final synchronized int lastIndexOf(Object elem,
                                          int index)
```

Searches backwards for the specified object, starting from the specified position and returns an index to it.

Parameters:

elem – the desired element

index – the index where to start searching

Returns:

the index of the element, or –1 if it was not found.

elementAt

```
public final synchronized Object elementAt(int index)
```

Returns the element at the specified index.

Parameters:

index – the index of the desired element

Throws: ArrayIndexOutOfBoundsException

If an invalid index was given.

firstElement

```
public final synchronized Object firstElement()
```

Returns the first element of the sequence.

Throws: NoSuchElementException

If the sequence is empty.

lastElement

```
public final synchronized Object lastElement()
```

Returns the last element of the sequence.

Throws: NoSuchElementException

If the sequence is empty.

setElementAt

```
public final synchronized void setElementAt(Object obj,
                                             int index)
```

Sets the element at the specified index to be the specified object. The previous element at that position is discarded.

Parameters:

obj – what the element is to be set to

index – the specified index

Throws: ArrayIndexOutOfBoundsException

If the index was invalid.

removeElementAt

```
public final synchronized void removeElementAt(int index)
```

Deletes the element at the specified index. Elements with an index greater than the current index are moved down.

Parameters:

index – the element to remove

Throws: ArrayIndexOutOfBoundsException

If the index was invalid.

insertElementAt

```
public final synchronized void insertElementAt(Object obj,
                                                int index)
```

Inserts the specified object as an element at the specified index. Elements with an index greater or equal to the current index are shifted up.

Parameters:

obj – the element to insert

index – where to insert the new element

Throws: ArrayIndexOutOfBoundsException

If the index was invalid.

addElement

```
public final synchronized void addElement(Object obj)
```

Adds the specified object as the last element of the vector.

Parameters:

obj – the element to be added

removeElement

```
public final synchronized boolean removeElement(Object obj)
```

Removes the element from the vector. If the object occurs more than once, only the first is removed. If the object is not an element, returns false.

Parameters:

obj – the element to be removed

Returns:

true if the element was actually removed; false otherwise.

removeAllElements

```
public final synchronized void removeAllElements()
```

Removes all elements of the vector. The vector becomes empty.

clone

```
public synchronized Object clone()
```

Clones this vector. The elements are **not** cloned.

Overrides:

clone in class Object

toString

```
public final synchronized String toString()
```

Converts the vector to a string. Useful for debugging.

Overrides:

toString in class Object