---

package java.lang

Interface Index

- Cloneable
- Runnable

Class Index

- Boolean
- Character
- Class
- ClassLoader
- Compiler
- Double
- Float
- Integer
- Long
- Math
- Number
- Object
- Process
- Runtime
- SecurityManager
- String
- StringBuffer
- System
- Thread
- ThreadGroup
- Throwable

Exception Index

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- ClassCastException
- ClassNotFoundException

- CloneNotSupportedException
- Exception
- IllegalAccessException
- IllegalArgumentException
- IllegalMonitorStateException
- IllegalThreadStateException
- IndexOutOfBoundsException
- InstantiationException
- InterruptedException
- NegativeArraySizeException
- NoSuchMethodException
- NullPointerException
- NumberFormatException
- RuntimeException
- SecurityException
- StringIndexOutOfBoundsException

- AbstractMethodError
- ClassCircularityError
- ClassFormatError
- Error
- IllegalAccessError
- IncompatibleClassChangeError
- InstantiationError
- InternalError
- LinkageError
- NoClassDefFoundError
- NoSuchFieldError
- NoSuchMethodError
- OutOfMemoryError
- StackOverflowError
- ThreadDeath
- UnknownError
- UnsatisfiedLinkError
- VerifyError
- VirtualMachineError

Class java.lang.AbstractMethodError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
                           |
                           +----java.lang.IncompatibleClassChangeError
                                   |
                                   +----java.lang.AbstractMethodError
```

---

public class **AbstractMethodError**
extends IncompatibleClassChangeError

Signals an attempt to call an abstract method.

Constructor Index

- **AbstractMethodError**()
  Constructs an AbstractMethodError with no detail message.
- **AbstractMethodError**(String)
  Constructs an AbstractMethodError with the specified detail message.

Constructors

AbstractMethodError

```
public AbstractMethodError()
```

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

AbstractMethodError

```
public AbstractMethodError(String s)
```

Constructs an AbstractMethodError with the specified detail message. A detail message is a

String that describes this particular exception.

Parameters:

s – the String that contains the detail message

Class java.lang.ArithmeticException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.ArithmeticException
```

---

public class **ArithmeticException**
extends RuntimeException

Signals that an exceptional arithmetic condition has occurred. For example, dividing by zero would invoke this class.

Constructor Index

- **ArithmeticException**()
  Constructs an ArithmeticException with no detail message.
- **ArithmeticException**(String)
  Constructs an ArithmeticException with the specified detail message.

Constructors

ArithmeticException

```
public ArithmeticException()
```

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

ArithmeticException

```
public ArithmeticException(String s)
```

Constructs an ArithmeticException with the specified detail message. A detail message is a

String that describes this particular exception.

Parameters:

s – the String that contains a detailed message

Class java.lang.ArrayIndexOutOfBoundsException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.IndexOutOfBoundsException
                                   |
                                   +----java.lang.ArrayIndexOutOfBoundsException
```

---

public class **ArrayIndexOutOfBoundsException**
extends IndexOutOfBoundsException

Signals that an invalid array index has been used.

Constructor Index

- **ArrayIndexOutOfBoundsException**()
  Constructs an ArrayIndexOutOfBoundsException with no detail message.
- **ArrayIndexOutOfBoundsException**(int)
  Constructs a new ArrayIndexOutOfBoundsException class initialized to the specific index.
- **ArrayIndexOutOfBoundsException**(String)
  Constructs an ArrayIndexOutOfBoundsException class with the specified detail message.

Constructors

ArrayIndexOutOfBoundsException

```
public ArrayIndexOutOfBoundsException()
```

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

ArrayIndexOutOfBoundsException

```
public ArrayIndexOutOfBoundsException(int index)
```

Constructs a new ArrayIndexOutOfBoundsException class initialized to the specific index.

Parameters:

index – the index where the error occurred

ArrayIndexOutOfBoundsException

```
public ArrayIndexOutOfBoundsException(String s)
```

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

Parameters:

s – the String containing a detail message

Class java.lang.ArrayStoreException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.ArrayStoreException
```

---

public class **ArrayStoreException**
extends RuntimeException

An attempt has been made to store the wrong type of Object to an array.

Constructor Index

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

Constructors

ArrayStoreException

```
public ArrayStoreException()
```

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

ArrayStoreException

```
public ArrayStoreException(String s)
```

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

Parameters:

s – the String containing a detail message

Class java.lang.Boolean

```
java.lang.Object
   |
   +----java.lang.Boolean
```

---

public final class **Boolean**
extends Object

The Boolean class provides an object wrapper for Boolean data values, and serves as a place for boolean−oriented operations. A wrapper is useful because most of Java's utility classes require the use of objects. Since booleans are not objects in Java, they need to be "wrapped" in a Boolean instance.

Variable Index

- **FALSE**
  Assigns this Boolean to be false.
- **MAX_VALUE**
  The maximum value a Character can have.
- **MIN_VALUE**
  The minimum value a Charater can have.
- **TRUE**
  Assigns this Boolean to be true.

Constructor Index

- **Boolean**(boolean)
  Constructs a Boolean object initialized to the specified boolean value.
- **Boolean**(String)
  Constructs a Boolean object initialized to the value specified by the String parameter.

Method Index

- **booleanValue**()
  Returns the value of this Boolean object as a boolean.
- **equals**(Object)

Compares this object against the specified object.

- **getBoolean**(String)
  Gets a Boolean from the properties.
- **hashCode**()
  Returns a hashcode for this Boolean.
- **toString**()
  Returns a new String object representing this Boolean's value.
- **valueOf**(String)
  Returns the boolean value represented by the specified String.

Variables

TRUE

```
public final static Boolean TRUE
```

Assigns this Boolean to be true.

FALSE

```
public final static Boolean FALSE
```

Assigns this Boolean to be false.

MIN_VALUE

```
public final static char MIN_VALUE
```

The minimum value a Charater can have. The lowest minimum value an Integer can have is ''.

MAX_VALUE

```
public final static char MAX_VALUE
```

The maximum value a Character can have. The greatest maximum value an Integer can have is 'ÿ'.

Constructors

Boolean

```
public Boolean(boolean value)
```

Constructs a Boolean object initialized to the specified boolean value.
Parameters:

value – the value of the boolean

Boolean

```
public Boolean(String s)
```

Constructs a Boolean object initialized to the value specified by the String parameter.
Parameters:
s – the String to be converted to a Boolean

Methods

booleanValue

```
public boolean booleanValue()
```

Returns the value of this Boolean object as a boolean.

valueOf

```
public static Boolean valueOf(String s)
```

Returns the boolean value represented by the specified String.
Parameters:
s – the String to be parsed

toString

```
public String toString()
```

Returns a new String object representing this Boolean's value.
Overrides:
toString in class Object

hashCode

```
public int hashCode()
```

Returns a hashcode for this Boolean.
Overrides:
hashCode in class Object

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

getBoolean

```
public static boolean getBoolean(String name)
```

Gets a Boolean from the properties.

Parameters:

name – the property name.

Class java.lang.Character

```
java.lang.Object
   |
   +----java.lang.Character
```

---

public final class **Character**
extends Object

The Character class provides an object wrapper for Character data values and serves as a place for character−oriented operations. A wrapper is useful because most of Java's utility classes require the use of objects. Since characters are not objects in Java, they need to be "wrapped" in a Character instance.

Variable Index

- **MAX_RADIX**
  The maximum radix available for conversion to and from Strings.
- **MIN_RADIX**
  The minimum radix available for conversion to and from Strings.

Constructor Index

- **Character**(char)
  Constructs a Character object with the specified value.

Method Index

- **charValue**()
  Returns the value of this Character object.
- **digit**(char, int)
  Returns the numeric value of the character digit using the specified radix.
- **equals**(Object)
  Compares this object against the specified object.
- **forDigit**(int, int)
  Returns the character value for the specified digit in the specified radix.

- **hashCode**()
  Returns a hashcode for this Character.
- **isDigit**(char)
  Determines if the specified character is a ISO−LATIN−1 digit.
- **isLowerCase**(char)
  Determines if the specified character is ISO−LATIN−1 lower case.
- **isSpace**(char)
  Determines if the specified character is ISO−LATIN−1 white space according to Java.
- **isUpperCase**(char)
  Determines if the specified character is ISO−LATIN−1 upper case.
- **toLowerCase**(char)
  Returns the lower case character value of the specified ISO−LATIN−1 character.
- **toString**()
  Returns a String object representing this character's value.
- **toUpperCase**(char)
  Returns the upper case character value of the specified ISO−LATIN−1 character.

MIN_RADIX

```
public final static int MIN_RADIX
```

The minimum radix available for conversion to and from Strings. The lowest minimum value that a radix can be is 2.

See Also:

toString

MAX_RADIX

```
public final static int MAX_RADIX
```

The maximum radix available for conversion to and from Strings. The largest maximum value that a radix can have is 36.

See Also:

toString

Character

```
public Character(char value)
```

Constructs a Character object with the specified value.

Parameters:

value – value of this Character object

Methods

isLowerCase

```
public static boolean isLowerCase(char ch)
```

Determines if the specified character is ISO–LATIN–1 lower case.

Parameters:

ch – the character to be tested

Returns:

true if the character is lower case; false otherwise.

isUpperCase

```
public static boolean isUpperCase(char ch)
```

Determines if the specified character is ISO–LATIN–1 upper case.

Parameters:

ch – the character to be tested

Returns:

true if the character is upper case; false otherwise.

isDigit

```
public static boolean isDigit(char ch)
```

Determines if the specified character is a ISO–LATIN–1 digit.

Parameters:

ch – the character to be tested

Returns:

true if this character is a digit; false otherwise.

isSpace

```
public static boolean isSpace(char ch)
```

Determines if the specified character is ISO–LATIN–1 white space according to Java.

Parameters:

ch – the character to be tested

Returns:

true if the character is white space; false otherwise.

toLowerCase

```
public static char toLowerCase(char ch)
```

Returns the lower case character value of the specified ISO−LATIN−1 character. Characters that are not upper case letters are returned unmodified.

Parameters:

ch − the character to be converted

toUpperCase

```
public static char toUpperCase(char ch)
```

Returns the upper case character value of the specified ISO−LATIN−1 character. Characters that are not lower case letters are returned unmodified. Note that German ess−zed and latin small letter y diaeresis have no corresponding upper case letters, even though they are lower case. There is a capital y diaeresis, but not in ISO−LATIN−1...

Parameters:

ch − the character to be converted

digit

```
public static int digit(char ch,
                        int radix)
```

Returns the numeric value of the character digit using the specified radix. If the character is not a valid digit, it returns −1.

Parameters:

ch − the character to be converted

radix − the radix

forDigit

```
public static char forDigit(int digit,
                            int radix)
```

Returns the character value for the specified digit in the specified radix. If the digit is not valid in the radix, the 0 character is returned.

Parameters:

digit − the digit chosen by the character value

radix − the radix containing the digit

charValue

```
public char charValue()
```

Returns the value of this Character object.

hashCode

```
public int hashCode()
```

Returns a hashcode for this Character.

Overrides:

hashCode in class Object

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

toString

```
public String toString()
```

Returns a String object representing this character's value.

Overrides:

toString in class Object

Class java.lang.Class

```
java.lang.Object
   |
   +----java.lang.Class
```

---

public final class **Class**
extends Object

Class objects contain runtime representations of classes. Every object in the system is an instance of some Class, and for each Class there is one of these descriptor objects. A Class descriptor is not modifiable at runtime.

The following example uses a Class object to print the Class name of an object:

```
void printClassName(Object obj) {
    System.out.println("The class of " + obj +
                       " is " + obj.getClass().getName());
}
```

Method Index

- **forName**(String)
  Returns the runtime Class descriptor for the specified Class.
- **getClassLoader**()
  Returns the Class loader of this Class.
- **getInterfaces**()
  Returns the interfaces of this Class.
- **getName**()
  Returns the name of this Class.
- **getSuperclass**()
  Returns the superclass of this Class.
- **isInterface**()
  Returns a boolean indicating whether or not this Class is an interface.
- **newInstance**()
  Creates a new instance of this Class.
- **toString**()
  Returns the name of this Class or this interface.

Methods

forName

```
public static Class forName(String className) throws ClassNotFoundException
```

Returns the runtime Class descriptor for the specified Class. For example, the following code fragment returns the runtime Class descriptor for the Class named java.lang.Thread:

```
Class t = Class.forName("java.lang.Thread")
```

Parameters:
className – the fully qualified name of the desired Class
Throws: ClassNotFoundException
If the Class could not be found.

newInstance

```
public Object newInstance() throws InstantiationException, IllegalAccessException
```

Creates a new instance of this Class.
Returns:
the new instance of this Class.
Throws: InstantiationException
If you try to instantiate an abstract class or an interface, or if the instantiation fails for some other reason.
Throws: IllegalAccessException
If the class or initializer is not accessible.

getName

```
public String getName()
```

Returns the name of this Class.

getSuperclass

```
public Class getSuperclass()
```

Returns the superclass of this Class.

getInterfaces

```
public Class[] getInterfaces()
```

Returns the interfaces of this Class. An array of length 0 is returned if this Class implements no interfaces.

getClassLoader

```
public ClassLoader getClassLoader()
```

Returns the Class loader of this Class. Returns null if this Class does not have a Class loader.

See Also:

ClassLoader

isInterface

```
public boolean isInterface()
```

Returns a boolean indicating whether or not this Class is an interface.

toString

```
public String toString()
```

Returns the name of this Class or this interface. The word "class" is prepended if it is a Class; the word "interface" is prepended if it is an interface.

Overrides:

toString in class Object

Class java.lang.ClassCastException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.ClassCastException
```

---

public class **ClassCastException**
extends RuntimeException

Signals that an invalid cast has occurred.

Constructor Index

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

Constructors

ClassCastException

```
public ClassCastException()
```

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

ClassCastException

```
public ClassCastException(String s)
```

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

Parameters:

s – the String containing a detail message

Class java.lang.ClassCircularityError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
                           |
                           +----java.lang.ClassCircularityError
```

---

public class **ClassCircularityError**
extends LinkageError

Signals that a circularity has been detected when initializing a class.

Constructor Index

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

Constructors

ClassCircularityError

```
public ClassCircularityError()
```

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

ClassCircularityError

```
public ClassCircularityError(String s)
```

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

Parameters:

s – the detail message

Class java.lang.ClassFormatError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
                           |
                           +----java.lang.ClassFormatError
```

---

public class **ClassFormatError**
extends LinkageError

Signals an invalid file format has occurred.

Constructor Index

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

Constructors

ClassFormatError

```
public ClassFormatError()
```

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

ClassFormatError

```
public ClassFormatError(String s)
```

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

Parameters:

s – the String containing the detail message

Class java.lang.ClassLoader

```
java.lang.Object
   |
   +----java.lang.ClassLoader
```

---

public class **ClassLoader**
extends Object

ClassLoader is an abstract Class that can be used to define a policy for loading Java classes into the runtime environment. By default, the runtime system loads classes that originate as files by reading them from the directory defined by the `CLASSPATH` environment variable (this is platform dependent). The default mechanism does not involve a Class loader.

However, some classes may not originate from a file; they could be loaded from some other source, e.g., the network. Classes loaded from the network are an array of bytes. A ClassLoader can be used to tell the runtime system to convert an array of bytes into an instance of class Class. This conversion information is passed to the runtime using the defineClass() method.

Classes that are created through the defineClass() mechanism can reference other classes by name. To resolve those names, the runtime system calls the ClassLoader that originally created the Class. The runtime system calls the abstract method loadClass() to load the referenced classes.

```
ClassLoader loader = new NetworkClassLoader(host, port);
Object main = loader.loadClass("Main").newInstance();
....
```

The NetworkClassLoader subclass must define the method loadClass() to load a Class from the network. Once it has downloaded the bytes that make up the Class it should use the method defineClass() to create a Class instance. A sample implementation could be:

```
class NetworkClassLoader {
    String host;
    int port;
    Hashtable cache = new Hashtable();
    private byte loadClassData(String name)[] {
        // load the class data from the connection
        ...
    }
    public synchronized Class loadClass(String name) {
        Class c = cache.get(name);
        if (c == null) {
            byte data[] = loadClassData(name);
            cache.put(name, defineClass(data, 0, data.length));
        }
        return c;
```

```
        }
    }
```

See Also:

Class

Constructor Index

- **ClassLoader**()
  Constructs a new Class loader and initializes it.

Method Index

- **defineClass**(byte[], int, int)
  Converts an array of bytes to an instance of class Class.
- **findSystemClass**(String)
  Loads a system Class.
- **loadClass**(String, boolean)
  Resolves the specified name to a Class.
- **resolveClass**(Class)
  Resolves classes referenced by this Class.

Constructors

ClassLoader

```
protected ClassLoader()
```

Constructs a new Class loader and initializes it.

Methods

loadClass

```
protected abstract Class loadClass(String name,
                                   boolean resolve) throws ClassNotFoundException
```

Resolves the specified name to a Class. The method loadClass() is called by the virtual machine. As an abstract method, loadClass() must be defined in a subclass of ClassLoader. By

using a Hashtable, you can avoid loading the same Class more than once.

Parameters:

name – the name of the desired Class

resolve – true if the Class needs to be resolved

Returns:

the resulting Class, or null if it was not found.

Throws: ClassNotFoundException

Cannot find a definition for the class

See Also:

Hashtable

defineClass

```
protected final Class defineClass(byte data[],
                                  int offset,
                                  int length)
```

Converts an array of bytes to an instance of class Class. Before the Class can be used it must be resolved.

Parameters:

data – the bytes that make up the Class

offset – the start offset of the Class data

length – the length of the Class data

Returns:

the Class object which was created from the data.

Throws: ClassFormatError

If the data does not contain a valid Class.

See Also:

loadClass, resolveClass

resolveClass

```
protected final void resolveClass(Class c)
```

Resolves classes referenced by this Class. This must be done before the Class can be used. Class names referenced by the resulting Class are resolved by calling loadClass().

Parameters:

c – the Class to be resolved

See Also:

defineClass

findSystemClass

```
protected final Class findSystemClass(String name) throws ClassNotFoundException
```

Loads a system Class. A system Class is a class with the primordial Class loader (which is null).

Parameters:

name – the name of the system Class

Throws: NoClassDefFoundError
If the Class is not found.
Throws: ClassNotFoundException
Cannot find a definition for the class

Class java.lang.ClassNotFoundException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.ClassNotFoundException
```

---

public class **ClassNotFoundException**
extends Exception

Signals that a class could not be found.

Constructor Index

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

Constructors

ClassNotFoundException

```
public ClassNotFoundException()
```

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

ClassNotFoundException

```
public ClassNotFoundException(String s)
```

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

Parameters:

s – the detail message

Class java.lang.CloneNotSupportedException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.CloneNotSupportedException
```

---

public class **CloneNotSupportedException**
extends Exception

Signals that an attempt has been made to clone an object that does not want to be cloned.

Constructor Index

- **CloneNotSupportedException**()
  Constructs an CloneNotSupportedException with no detail message.
- **CloneNotSupportedException**(String)
  Constructs an CloneNotSupportedException with the specified detail message.

Constructors

CloneNotSupportedException

```
public CloneNotSupportedException()
```

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

CloneNotSupportedException

```
public CloneNotSupportedException(String s)
```

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

Parameters:

s – the detail message

Interface java.lang.Cloneable

public interface **Cloneable**
extends Object

An interface indicating that this object doesn't mind being copied or cloned.

Class java.lang.Compiler

```
java.lang.Object
    |
    +----java.lang.Compiler
```

---

public final class **Compiler**
extends Object

Method Index

- **command**(Object)
- **compileClass**(Class)
- **compileClasses**(String)
- **disable**()
- **enable**()

Methods

compileClass

```
public static boolean compileClass(Class clazz)
```

compileClasses

```
public static boolean compileClasses(String string)
```

command

```
public static Object command(Object any)
```

enable

```
public static void enable()
```

disable

```
public static void disable()
```

Class java.lang.Double

```
java.lang.Object
   |
   +----java.lang.Number
           |
           +----java.lang.Double
```

---

public final class **Double**
extends Number

The Double class provides an object wrapper for Double data values and serves as a place for double−oriented operations. A wrapper is useful because most of Java's utility classes require the use of objects. Since doubles are not objects in Java, they need to be "wrapped" in a Double instance.

Variable Index

- **MAX_VALUE**
  The maximum value a double can have.
- **MIN_VALUE**
  The minimum value a double can have.
- **NEGATIVE_INFINITY**
  Negative infinity.
- **NaN**
  Not−a−Number.
- **POSITIVE_INFINITY**
  Positive infinity.

Constructor Index

- **Double**(double)
  Constructs a Double wrapper for the specified double value.
- **Double**(String)
  Constructs a Double object initialized to the value specified by the String parameter.

Method Index

- **doubleToLongBits**(double)
  Returns the bit represention of a double−float value
- **doubleValue**()
  Returns the double value of this Double.
- **equals**(Object)
  Compares this object against the specified object.
- **floatValue**()
  Returns the float value of this Double.
- **hashCode**()
  Returns a hashcode for this Double.
- **intValue**()
  Returns the integer value of this Double (by casting to an int).
- **isInfinite**(double)
  Returns true if the specified number is infinitely large in magnitude.
- **isInfinite**()
  Returns true if this Double value is infinitely large in magnitude.
- **isNaN**(double)
  Returns true if the specified number is the special Not−a−Number (NaN) value.
- **isNaN**()
  Returns true if this Double value is the special Not−a−Number (NaN) value.
- **longBitsToDouble**(long)
  Returns the double−float corresponding to a given bit represention.
- **longValue**()
  Returns the long value of this Double (by casting to a long).
- **toString**(double)
  Returns a String representation for the specified double value.
- **toString**()
  Returns a String representation of this Double object.
- **valueOf**(String)
  Returns a new Double value initialized to the value represented by the specified String.

Variables

POSITIVE_INFINITY

```
public final static double POSITIVE_INFINITY
```

Positive infinity.

NEGATIVE_INFINITY

```
public final static double NEGATIVE_INFINITY
```

Negative infinity.

NaN

```
public final static double NaN
```

Not−a−Number. *Note: is not equal to anything, including itself*

MAX_VALUE

```
public final static double MAX_VALUE
```

The maximum value a double can have. The greatest maximum value that a double can have is 1.79769313486231570e+308d.

MIN_VALUE

```
public final static double MIN_VALUE
```

The minimum value a double can have. The lowest minimum value that a double can have is 4.94065645841246544e−324d.

Constructors

Double

```
public Double(double value)
```

Constructs a Double wrapper for the specified double value.
Parameters:
value − the initial value of the double

Double

```
public Double(String s) throws NumberFormatException
```

Constructs a Double object initialized to the value specified by the String parameter.
Parameters:
s − the String to be converted to a Double
Throws: NumberFormatException
If the String does not contain a parsable number.

Methods

toString

```
public static String toString(double d)
```

Returns a String representation for the specified double value.

Parameters:

d – the double to be converted

valueOf

```
public static Double valueOf(String s) throws NumberFormatException
```

Returns a new Double value initialized to the value represented by the specified String.

Parameters:

s – the String to be parsed

Throws: NumberFormatException

If the String cannot be parsed.

isNaN

```
public static boolean isNaN(double v)
```

Returns true if the specified number is the special Not–a–Number (NaN) value.

Parameters:

v – the value to be tested

isInfinite

```
public static boolean isInfinite(double v)
```

Returns true if the specified number is infinitely large in magnitude.

Parameters:

v – the value to be tested

isNaN

```
public boolean isNaN()
```

Returns true if this Double value is the special Not–a–Number (NaN) value.

isInfinite

```
public boolean isInfinite()
```

Returns true if this Double value is infinitely large in magnitude.

toString

```
public String toString()
```

Returns a String representation of this Double object.

Overrides:
toString in class Object

intValue

```
public int intValue()
```

Returns the integer value of this Double (by casting to an int).
Overrides:
intValue in class Number

longValue

```
public long longValue()
```

Returns the long value of this Double (by casting to a long).
Overrides:
longValue in class Number

floatValue

```
public float floatValue()
```

Returns the float value of this Double.
Overrides:
floatValue in class Number

doubleValue

```
public double doubleValue()
```

Returns the double value of this Double.
Overrides:
doubleValue in class Number

hashCode

```
public int hashCode()
```

Returns a hashcode for this Double.
Overrides:
hashCode in class Object

equals

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

Compares this object against the specified object.

Note: To be useful in hashtables this method considers two NaN double values to be equal. This is not according to IEEE specification

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object

doubleToLongBits

```
public static long doubleToLongBits(double value)
```

Returns the bit represention of a double–float value

longBitsToDouble

```
public static double longBitsToDouble(long bits)
```

Returns the double–float corresponding to a given bit represention.

Class java.lang.Error

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
```

---

public class **Error**
extends Throwable

Error is a subtype of Throwable for abnormal events that should not occur. Do not try to catch Error's unless you really know what you're doing.

Constructor Index

- **Error**()
  Constructs an Error with no specified detail message.
- **Error**(String)
  Constructs an Error with the specified detail message.

Constructors

Error

```
public Error()
```

Constructs an Error with no specified detail message. A detail message is a String that describes this particular error.

Error

```
public Error(String s)
```

Constructs an Error with the specified detail message. A detail message is a String that describes this particular error

Parameters:

s – the detail message

Class java.lang.Exception

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
```

---

public class **Exception**
extends Throwable

Exception are a form of Throwable that normal programs may wish to try and catch.

Constructor Index

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

Constructors

Exception

```
public Exception()
```

Constructs an Exception with no specified detail message. A detail message is a String that describes this particular exception.

Exception

```
public Exception(String s)
```

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

Parameters:

s – the detail message

Class java.lang.Float

```
java.lang.Object
    |
    +----java.lang.Number
             |
             +----java.lang.Float
```

---

public final class **Float**
extends Number

The Float class provides an object wrapper for Float data values, and serves as a place for float−oriented operations. A wrapper is useful because most of Java's utility classes require the use of objects. Since floats are not objects in Java, they need to be "wrapped" in a Float instance.

Variable Index

- **MAX_VALUE**
  The maximum value a float can have.
- **MIN_VALUE**
  The minimum value a float can have.
- **NEGATIVE_INFINITY**
  Negative infinity.
- **NaN**
  Not−a−Number.
- **POSITIVE_INFINITY**
  Positive infinity.

Constructor Index

- **Float**(float)
  Constructs a Float wrapper for the specified float value.
- **Float**(double)
  Constructs a Float wrapper for the specified double value.
- **Float**(String)
  Constructs a Float object initialized to the value specified by the String parameter.

Method Index

- **doubleValue**()
  Returns the double value of this Float.
- **equals**(Object)
  Compares this object against some other object.
- **floatToIntBits**(float)
  Returns the bit represention of a single−float value
- **floatValue**()
  Returns the float value of this Float object.
- **hashCode**()
  Returns a hashcode for this Float.
- **intBitsToFloat**(int)
  Returns the single−float corresponding to a given bit represention.
- **intValue**()
  Returns the integer value of this Float (by casting to an int).
- **isInfinite**(float)
  Returns true if the specified number is infinitely large in magnitude.
- **isInfinite**()
  Returns true if this Float value is infinitely large in magnitude.
- **isNaN**(float)
  Returns true if the specified number is the special Not−a−Number (NaN) value.
- **isNaN**()
  Returns true if this Float value is Not−a−Number (NaN).
- **longValue**()
  Returns the long value of this Float (by casting to a long).
- **toString**(float)
  Returns a String representation for the specified float value.
- **toString**()
  Returns a String representation of this Float object.
- **valueOf**(String)
  Returns the floating point value represented by the specified String.

Variables

POSITIVE_INFINITY

```
public final static float POSITIVE_INFINITY
```

Positive infinity.

NEGATIVE_INFINITY

```
public final static float NEGATIVE_INFINITY
```

Negative infinity.

NaN

```
public final static float NaN
```

Not–a–Number. *Note: is not equal to anything, including itself*

MAX_VALUE

```
public final static float MAX_VALUE
```

The maximum value a float can have. The largest maximum value possible is 3.40282346638528860e+38.

MIN_VALUE

```
public final static float MIN_VALUE
```

The minimum value a float can have. The lowest minimum value possible is 1.40129846432481707e–45.

Constructors

Float

```
public Float(float value)
```

Constructs a Float wrapper for the specified float value.
Parameters:
value – the value of the Float

Float

```
public Float(double value)
```

Constructs a Float wrapper for the specified double value.
Parameters:
value – the value of the Float

Float

```
public Float(String s) throws NumberFormatException
```

Constructs a Float object initialized to the value specified by the String parameter.
Parameters:
s – the String to be converted to a Float
Throws: NumberFormatException
If the String does not contain a parsable number.

Methods

toString

```
public static String toString(float f)
```

Returns a String representation for the specified float value.

Parameters:

f – the float to be converted

valueOf

```
public static Float valueOf(String s) throws NumberFormatException
```

Returns the floating point value represented by the specified String.

Parameters:

s – the String to be parsed

Throws: NumberFormatException

If the String does not contain a parsable Float.

isNaN

```
public static boolean isNaN(float v)
```

Returns true if the specified number is the special Not–a–Number (NaN) value.

Parameters:

v – the value to be tested

isInfinite

```
public static boolean isInfinite(float v)
```

Returns true if the specified number is infinitely large in magnitude.

Parameters:

v – the value to be tested

isNaN

```
public boolean isNaN()
```

Returns true if this Float value is Not–a–Number (NaN).

isInfinite

```
public boolean isInfinite()
```

Returns true if this Float value is infinitely large in magnitude.

toString

```
public String toString()
```

Returns a String representation of this Float object.
Overrides:
toString in class Object

intValue

```
public int intValue()
```

Returns the integer value of this Float (by casting to an int).
Overrides:
intValue in class Number

longValue

```
public long longValue()
```

Returns the long value of this Float (by casting to a long).
Overrides:
longValue in class Number

floatValue

```
public float floatValue()
```

Returns the float value of this Float object.
Overrides:
floatValue in class Number

doubleValue

```
public double doubleValue()
```

Returns the double value of this Float.
Overrides:
doubleValue in class Number

hashCode

```
public int hashCode()
```

Returns a hashcode for this Float.
Overrides:
hashCode in class Object

equals

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

Compares this object against some other object.

Note: To be useful in hashtables this method considers two Nan floating point values to be equal. This is not according to IEEE specification

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object

floatToIntBits

```
public static int floatToIntBits(float value)
```

Returns the bit represention of a single–float value

intBitsToFloat

```
public static float intBitsToFloat(int bits)
```

Returns the single–float corresponding to a given bit represention.

Class java.lang.IllegalAccessError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
                           |
                           +----java.lang.IncompatibleClassChangeError
                                   |
                                   +----java.lang.IllegalAccessError
```

---

public class **IllegalAccessError**
extends IncompatibleClassChangeError

Signals that an illegal access exception has occurred.

Constructor Index

- **IllegalAccessError**()
  Constructs an IllegalAccessError with no detail message.
- **IllegalAccessError**(String)
  Constructs an IllegalAccessError with the specified detail message.

Constructors

IllegalAccessError

```
public IllegalAccessError()
```

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

IllegalAccessError

```
public IllegalAccessError(String s)
```

Constructs an IllegalAccessError with the specified detail message. A detail message is a String

that describes this particular exception.

Parameters:

s – the detail message

Class java.lang.IllegalAccessException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.IllegalAccessException
```

---

public class **IllegalAccessException**
extends Exception

Signals that a particular method could not be found.

Constructor Index

- **IllegalAccessException**()
  Constructs a IllegalAccessException without a detail message.
- **IllegalAccessException**(String)
  Constructs a IllegalAccessException with a detail message.

Constructors

IllegalAccessException

```
public IllegalAccessException()
```

Constructs a IllegalAccessException without a detail message. A detail message is a String that describes this particular exception.

IllegalAccessException

```
public IllegalAccessException(String s)
```

Constructs a IllegalAccessException with a detail message. A detail message is a String that describes this particular exception.

Parameters:

s – the detail message

Class java.lang.IllegalArgumentException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.IllegalArgumentException
```

---

public class **IllegalArgumentException**
extends RuntimeException

Signals that an illegal argument exception has occurred.

See Also:
setPriority

Constructor Index

- **IllegalArgumentException**()
  Constructs an IllegalArgumentException with no detail message.
- **IllegalArgumentException**(String)
  Constructs an IllegalArgumentException with the specified detail message.

Constructors

IllegalArgumentException

```
public IllegalArgumentException()
```

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

IllegalArgumentException

```
public IllegalArgumentException(String s)
```

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

Parameters:

s – the detail message

Class java.lang.IllegalMonitorStateException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.IllegalMonitorStateException
```

---

public class **IllegalMonitorStateException**
extends RuntimeException

Signals that a monitor operation has been attempted when the monitor is in an invalid state. For example, trying to notify a monitor that you do not own would invoke this class.

Constructor Index

- **IllegalMonitorStateException**()
  Constructs an IllegalMonitorStateException with no detail message.
- **IllegalMonitorStateException**(String)
  Constructs an IllegalMonitorStateException with the specified detail message.

Constructors

IllegalMonitorStateException

```
public IllegalMonitorStateException()
```

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

IllegalMonitorStateException

```
public IllegalMonitorStateException(String s)
```

Constructs an IllegalMonitorStateException with the specified detail message. A detail message

is a String that describes this particular exception.

Parameters:

s – the String that contains a detailed message

Class java.lang.IllegalThreadStateException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.IllegalArgumentException
                                   |
                                   +----java.lang.IllegalThreadStateException
```

---

public class **IllegalThreadStateException**
extends IllegalArgumentException

Exception indicating that a thread is not in the proper state for the requested operation.

See Also:
suspend, resume

Constructor Index

- **IllegalThreadStateException**()
  Constructs an IllegalThreadStateException with no detail message.
- **IllegalThreadStateException**(String)
  Constructs an IllegalThreadStateException with the specified detail message.

Constructors

IllegalThreadStateException

```
public IllegalThreadStateException()
```

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

IllegalThreadStateException

```
public IllegalThreadStateException(String s)
```

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

Parameters:

s – the detail message

Class java.lang.IncompatibleClassChangeError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
                           |
                           +----java.lang.IncompatibleClassChangeError
```

---

public class **IncompatibleClassChangeError**
extends LinkageError

Signals that an incompatible class change has occurred.

Constructor Index

- **IncompatibleClassChangeError**()
  Constructs an IncompatibleClassChangeError with no detail message.
- **IncompatibleClassChangeError**(String)
  Constructs an IncompatibleClassChangeError with the specified detail message.

Constructors

IncompatibleClassChangeError

```
public IncompatibleClassChangeError()
```

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

IncompatibleClassChangeError

```
public IncompatibleClassChangeError(String s)
```

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

Parameters:

s – the detail message

Class java.lang.IndexOutOfBoundsException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.IndexOutOfBoundsException
```

---

public class **IndexOutOfBoundsException**
extends RuntimeException

Signals that an index of some sort is out of bounds.

Constructor Index

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

Constructors

IndexOutOfBoundsException

```
public IndexOutOfBoundsException()
```

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

IndexOutOfBoundsException

```
public IndexOutOfBoundsException(String s)
```

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

Parameters:

s – the detail message

Class java.lang.InstantiationError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
                           |
                           +----java.lang.IncompatibleClassChangeError
                                   |
                                   +----java.lang.InstantiationError
```

---

public class **InstantiationError**
extends IncompatibleClassChangeError

Signals that the interpreter has tried to instantiate an abstract class or an interface.

Constructor Index

- **InstantiationError**()
  Constructs an InstantiationError with no detail message.
- **InstantiationError**(String)
  Constructs an InstantiationError with the specified detail message.

Constructors

InstantiationError

```
public InstantiationError()
```

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

InstantiationError

```
public InstantiationError(String s)
```

Constructs an InstantiationError with the specified detail message. A detail message is a String

that describes this particular exception.

Parameters:

s – the String that contains the detail message

Class java.lang.InstantiationException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.InstantiationException
```

---

public class **InstantiationException**
extends Exception

Signals that an attempt has been made to instantiate an abstract class or an interface.

Constructor Index

- **InstantiationException**()
  Constructs an InstantiationException with no detail message.
- **InstantiationException**(String)
  Constructs an InstantiationException with the specified detail message.

Constructors

InstantiationException

```
public InstantiationException()
```

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

InstantiationException

```
public InstantiationException(String s)
```

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

Parameters:

s – the String containing a detail message

Class java.lang.Integer

```
java.lang.Object
   |
   +----java.lang.Number
           |
           +----java.lang.Integer
```

---

public final class **Integer**
extends Number

The Integer class is a wrapper for integer values. In Java, integers are not objects and most of the Java utility classes require the use of objects. Thus, if you needed to store an integer in a hashtable, you would have to "wrap" an Integer instance around it.

Variable Index

- **MAX_VALUE**
  The maximum value an Integer can have.
- **MIN_VALUE**
  The minimum value an Integer can have.

Constructor Index

- **Integer**(int)
  Constructs an Integer object initialized to the specified int value.
- **Integer**(String)
  Constructs an Integer object initialized to the value specified by the String parameter.

Method Index

- **doubleValue**()
  Returns the value of this Integer as a double.
- **equals**(Object)
  Compares this object to the specified object.
- **floatValue**()
  Returns the value of this Integer as a float.

- **getInteger**(String)
  Gets an Integer property.
- **getInteger**(String, int)
  Gets an Integer property.
- **getInteger**(String, Integer)
  Gets an Integer property.
- **hashCode**()
  Returns a hashcode for this Integer.
- **intValue**()
  Returns the value of this Integer as an int.
- **longValue**()
  Returns the value of this Integer as a long.
- **parseInt**(String, int)
  Assuming the specified String represents an integer, returns that integer's value.
- **parseInt**(String)
  Assuming the specified String represents an integer, returns that integer's value.
- **toString**(int, int)
  Returns a new String object representing the specified integer in the specified radix.
- **toString**(int)
  Returns a new String object representing the specified integer.
- **toString**()
  Returns a String object representing this Integer's value.
- **valueOf**(String, int)
  Assuming the specified String represents an integer, returns a new Integer object initialized to that value.
- **valueOf**(String)
  Assuming the specified String represents an integer, returns a new Integer object initialized to that value.

MIN_VALUE

```
public final static int MIN_VALUE
```

The minimum value an Integer can have. The lowest minimum value an Integer can have is 0x80000000.

MAX_VALUE

```
public final static int MAX_VALUE
```

The maximum value an Integer can have. The greatest maximum value an Integer can have is 0x7fffffff.

Constructors

Integer

```
public Integer(int value)
```

Constructs an Integer object initialized to the specified int value.

Parameters:

value – the initial value of the Integer

Integer

```
public Integer(String s) throws NumberFormatException
```

Constructs an Integer object initialized to the value specified by the String parameter. The radix is assumed to be 10.

Parameters:

s – the String to be converted to an Integer

Throws: NumberFormatException

If the String does not contain a parsable integer.

Methods

toString

```
public static String toString(int i,
                              int radix)
```

Returns a new String object representing the specified integer in the specified radix.

Parameters:

i – the integer to be converted

radix – the radix

See Also:

MIN_RADIX, MAX_RADIX

toString

```
public static String toString(int i)
```

Returns a new String object representing the specified integer. The radix is assumed to be 10.

Parameters:

i – the integer to be converted

parseInt

```
public static int parseInt(String s,
```

```
                                int radix) throws NumberFormatException
```

Assuming the specified String represents an integer, returns that integer's value. Throws an exception if the String cannot be parsed as an int.

Parameters:

s – the String containing the integer

radix – the radix to be used

Throws: NumberFormatException

If the String does not contain a parsable integer.

parseInt

```
public static int parseInt(String s) throws NumberFormatException
```

Assuming the specified String represents an integer, returns that integer's value. Throws an exception if the String cannot be parsed as an int. The radix is assumed to be 10.

Parameters:

s – the String containing the integer

Throws: NumberFormatException

If the string does not contain a parsable integer.

valueOf

```
public static Integer valueOf(String s,
                              int radix) throws NumberFormatException
```

Assuming the specified String represents an integer, returns a new Integer object initialized to that value. Throws an exception if the String cannot be parsed as an int.

Parameters:

s – the String containing the integer

radix – the radix to be used

Throws: NumberFormatException

If the String does not contain a parsable integer.

valueOf

```
public static Integer valueOf(String s) throws NumberFormatException
```

Assuming the specified String represents an integer, returns a new Integer object initialized to that value. Throws an exception if the String cannot be parsed as an int. The radix is assumed to be 10.

Parameters:

s – the String containing the integer

Throws: NumberFormatException

If the String does not contain a parsable integer.

intValue

```
public int intValue()
```

Returns the value of this Integer as an int.
Overrides:
intValue in class Number

longValue

```
public long longValue()
```

Returns the value of this Integer as a long.
Overrides:
longValue in class Number

floatValue

```
public float floatValue()
```

Returns the value of this Integer as a float.
Overrides:
floatValue in class Number

doubleValue

```
public double doubleValue()
```

Returns the value of this Integer as a double.
Overrides:
doubleValue in class Number

toString

```
public String toString()
```

Returns a String object representing this Integer's value.
Overrides:
toString in class Object

hashCode

```
public int hashCode()
```

Returns a hashcode for this Integer.
Overrides:
hashCode in class Object

equals

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

Compares this object to 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

getInteger

```
public static Integer getInteger(String nm)
```

Gets an Integer property. If the property does not exist, it will return 0.

Parameters:

nm – the property name

getInteger

```
public static Integer getInteger(String nm,
                                 int val)
```

Gets an Integer property. If the property does not exist, it will return val. Deals with Hexadecimal and octal numbers.

Parameters:

nm – the String name

val – the Integer value

getInteger

```
public static Integer getInteger(String nm,
                                 Integer val)
```

Gets an Integer property. If the property does not exist, it will return val. Deals with Hexadecimal and octal numbers.

Parameters:

nm – the property name

val – the integer value

Class java.lang.InternalError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.VirtualMachineError
                           |
                           +----java.lang.InternalError
```

---

public class **InternalError**
extends VirtualMachineError

Signals that an internal error has occurred.

Constructor Index

- **InternalError**()
  Constructs an InternalError with no detail message.
- **InternalError**(String)
  Constructs an InternalError with the specified detail message.

Constructors

InternalError

```
public InternalError()
```

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

InternalError

```
public InternalError(String s)
```

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

Parameters:

s – the detail message

Class java.lang.InterruptedException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.InterruptedException
```

---

public class **InterruptedException**
extends Exception

An exception indicated that some thread has interrupted this thread.

See Also:
interrupt, interrupted

Constructor Index

- **InterruptedException**()
  Constructs an InterruptedException with no detail message.
- **InterruptedException**(String)
  Constructs an InterruptedException with the specified detail message.

Constructors

InterruptedException

```
public InterruptedException()
```

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

InterruptedException

```
public InterruptedException(String s)
```

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

Parameters:

s – the detail message

Class java.lang.LinkageError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
```

---

public class **LinkageError**
extends Error

LinkageError and its subclasses indicate that a class has some dependency on another class; however the latter class has incompatibly changed after the compilation of the former class.

Constructor Index

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

Constructors

LinkageError

```
public LinkageError()
```

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

LinkageError

```
public LinkageError(String s)
```

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

Parameters:

s – the detail message

Class java.lang.Long

```
java.lang.Object
   |
   +----java.lang.Number
           |
           +----java.lang.Long
```

---

public final class **Long**
extends Number

The Long class provides an object wrapper for Long data values and serves as a place for long−oriented operations. A wrapper is useful because most of Java's utility classes require the use of objects. Since longs are not objects in Java, they need to be "wrapped" in a Long instance.

Variable Index

- **MAX_VALUE**
  The maximum value a Long can have.
- **MIN_VALUE**
  The minimum value a Long can have.

Constructor Index

- **Long**(long)
  Constructs a Long object initialized to the specified value.
- **Long**(String)
  Constructs a Long object initialized to the value specified by the String parameter.

Method Index

- **doubleValue**()
  Returns the value of this Long as a double.
- **equals**(Object)
  Compares this object against the specified object.
- **floatValue**()
  Returns the value of this Long as a float.

- **getLong**(String)
  Get a Long property.
- **getLong**(String, long)
  Get a Long property.
- **getLong**(String, Long)
  Get a Long property.
- **hashCode**()
  Computes a hashcode for this Long.
- **intValue**()
  Returns the value of this Long as an int.
- **longValue**()
  Returns the value of this Long as a long.
- **parseLong**(String, int)
  Assuming the specified String represents a long, returns that long's value.
- **parseLong**(String)
  Assuming the specified String represents a long, return that long's value.
- **toString**(long, int)
  Returns a new String object representing the specified long in the specified radix.
- **toString**(long)
  Returns a new String object representing the specified integer.
- **toString**()
  Returns a String object representing this Long's value.
- **valueOf**(String, int)
  Assuming the specified String represents a long, returns a new Long object initialized to that value.
- **valueOf**(String)
  Assuming the specified String represents a long, returns a new Long object initialized to that value.

MIN_VALUE

```
public final static long MIN_VALUE
```

The minimum value a Long can have. The lowest minimum value that a Long can have is 0x8000000000000000.

MAX_VALUE

```
public final static long MAX_VALUE
```

The maximum value a Long can have. The larget maximum value that a Long can have is 0x7fffffffffffffff.

Constructors

Long

```
public Long(long value)
```

Constructs a Long object initialized to the specified value.

Parameters:

value – the initial value of the Long

Long

```
public Long(String s) throws NumberFormatException
```

Constructs a Long object initialized to the value specified by the String parameter. The radix is assumed to be 10.

Parameters:

s – the String to be converted to a Long

Throws: NumberFormatException

If the String does not contain a parsable long.

Methods

toString

```
public static String toString(long i,
                              int radix)
```

Returns a new String object representing the specified long in the specified radix.

Parameters:

i – the long to be converted

radix – the radix

See Also:

MIN_RADIX, MAX_RADIX

toString

```
public static String toString(long i)
```

Returns a new String object representing the specified integer. The radix is assumed to be 10.

Parameters:

i – the long to be converted

parseLong

```
public static long parseLong(String s,
```

```
                                int radix) throws NumberFormatException
```

Assuming the specified String represents a long, returns that long's value. Throws an exception if the String cannot be parsed as a long.

Parameters:

s – the String containing the integer

radix – the radix to be used

Throws: NumberFormatException

If the String does not contain a parsable integer.

parseLong

```
public static long parseLong(String s) throws NumberFormatException
```

Assuming the specified String represents a long, return that long's value. Throws an exception if the String cannot be parsed as a long. The radix is assumed to be 10.

Parameters:

s – the String containing the long

Throws: NumberFormatException

If the string does not contain a parsable long.

valueOf

```
public static Long valueOf(String s,
                           int radix) throws NumberFormatException
```

Assuming the specified String represents a long, returns a new Long object initialized to that value. Throws an exception if the String cannot be parsed as a long.

Parameters:

s – the String containing the long.

radix – the radix to be used

Throws: NumberFormatException

If the String does not contain a parsable long.

valueOf

```
public static Long valueOf(String s) throws NumberFormatException
```

Assuming the specified String represents a long, returns a new Long object initialized to that value. Throws an exception if the String cannot be parsed as a long. The radix is assumed to be 10.

Parameters:

s – the String containing the long

Throws: NumberFormatException

If the String does not contain a parsable long.

intValue

```
public int intValue()
```

Returns the value of this Long as an int.
Overrides:
intValue in class Number

longValue

```
public long longValue()
```

Returns the value of this Long as a long.
Overrides:
longValue in class Number

floatValue

```
public float floatValue()
```

Returns the value of this Long as a float.
Overrides:
floatValue in class Number

doubleValue

```
public double doubleValue()
```

Returns the value of this Long as a double.
Overrides:
doubleValue in class Number

toString

```
public String toString()
```

Returns a String object representing this Long's value.
Overrides:
toString in class Object

hashCode

```
public int hashCode()
```

Computes a hashcode for this Long.
Overrides:
hashCode in class Object

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

getLong

```
public static Long getLong(String nm)
```

Get a Long property. If the property does not exist, it will return 0.

Parameters:

nm – the property name

getLong

```
public static Long getLong(String nm,
                           long val)
```

Get a Long property. If the property does not exist, it will return val. Deals with Hexadecimal and octal numbers.

Parameters:

nm – the String name

val – the Long value

getLong

```
public static Long getLong(String nm,
                           Long val)
```

Get a Long property. If the property does not exist, it will return val. Deals with Hexadecimal and octal numbers.

Parameters:

nm – the property name

val – the Long value

Class java.lang.Math

```
java.lang.Object
   |
   +----java.lang.Math
```

---

public final class **Math**
extends Object

The standard Math library. For the methods in this Class, error handling for out−of−range or immeasurable results are platform dependent. This class cannot be subclassed or instantiated because all methods and variables are static.

Variable Index

- **E**
  The float representation of the value E.
- **PI**
  The float representation of the value Pi.

Method Index

- **IEEEremainder**(double, double)
  Returns the remainder of f1 divided by f2 as defined by IEEE 754.
- **abs**(int)
  Returns the absolute integer value of a.
- **abs**(long)
  Returns the absolute long value of a.
- **abs**(float)
  Returns the absolute float value of a.
- **abs**(double)
  Returns the absolute double value of a.
- **acos**(double)
  Returns the arc cosine of a, in the range of 0.0 through Pi.
- **asin**(double)
  Returns the arc sine of a, in the range of −Pi/2 through Pi/2.
- **atan**(double)
  Returns the arc tangent of a, in the range of −Pi/2 through Pi/2.

- **atan2**(double, double)
  Converts rectangular coordinates (a, b) to polar (r, theta).
- **ceil**(double)
  Returns the "ceiling" or smallest whole number greater than or equal to a.
- **cos**(double)
  Returns the trigonometric cosine of an angle.
- **exp**(double)
  Returns the exponential number e(2.718...) raised to the power of a.
- **floor**(double)
  Returns the "floor" or largest whole number less than or equal to a.
- **log**(double)
  Returns the natural logarithm (base e) of a.
- **max**(int, int)
  Takes two int values, a and b, and returns the greater number of the two.
- **max**(long, long)
  Takes two long values, a and b, and returns the greater number of the two.
- **max**(float, float)
  Takes two float values, a and b, and returns the greater number of the two.
- **max**(double, double)
  Takes two double values, a and b, and returns the greater number of the two.
- **min**(int, int)
  Takes two integer values, a and b, and returns the smallest number of the two.
- **min**(long, long)
  Takes two long values, a and b, and returns the smallest number of the two.
- **min**(float, float)
  Takes two float values, a and b, and returns the smallest number of the two.
- **min**(double, double)
  Takes two double values, a and b, and returns the smallest number of the two.
- **pow**(double, double)
  Returns the number a raised to the power of b.
- **random**()
  Generates a random number between 0.0 and 1.0.
- **rint**(double)
  Converts a double value into an integral value in double format.
- **round**(float)
  Rounds off a float value by first adding 0.5 to it and then returning the largest integer that is less than or equal to this new value.
- **round**(double)
  Rounds off a double value by first adding 0.5 to it and then returning the largest integer that is less than or equal to this new value.
- **sin**(double)
  Returns the trigonometric sine of an angle.
- **sqrt**(double)
  Returns the square root of a.
- **tan**(double)
  Returns the trigonometric tangent of an angle.

Variables

E

```
public final static double E
```

The float representation of the value E. E is equivalent to 2.7182818284590452354f in Java.

PI

```
public final static double PI
```

The float representation of the value Pi. Pi is equivalent to 3.14159265358979323846f in Java.

Methods

sin

```
public static double sin(double a)
```

Returns the trigonometric sine of an angle.
Parameters:
a – an assigned angle that is measured in radians

cos

```
public static double cos(double a)
```

Returns the trigonometric cosine of an angle.
Parameters:
a – an assigned angle that is measured in radians

tan

```
public static double tan(double a)
```

Returns the trigonometric tangent of an angle.
Parameters:
a – an assigned angle that is measured in radians

asin

```
public static double asin(double a)
```

Returns the arc sine of a, in the range of –Pi/2 through Pi/2.

Parameters:

a – (–1.0) **acos**

```
public static double acos(double a)
```

Returns the arc cosine of a, in the range of 0.0 through Pi.

Parameters:

a – (–1.0) **atan**

```
public static double atan(double a)
```

Returns the arc tangent of a, in the range of –Pi/2 through Pi/2.

Parameters:

a – an assigned value

Returns:

the arc tangent of a.

exp

```
public static double exp(double a)
```

Returns the exponential number e(2.718...) raised to the power of a.

Parameters:

a – an assigned value

log

```
public static double log(double a) throws ArithmeticException
```

Returns the natural logarithm (base e) of a.

Parameters:

a – a is a number greater than 0.0

Throws: ArithmeticException

If a is less than 0.0 .

sqrt

```
public static double sqrt(double a) throws ArithmeticException
```

Returns the square root of a.

Parameters:

a – a is a number greater than or equal to 0.0

Throws: ArithmeticException

If a is a value less than 0.0 .

IEEEremainder

```
public static double IEEEremainder(double f1,
                                   double f2)
```

Returns the remainder of f1 divided by f2 as defined by IEEE 754.

Parameters:

f1 – the dividend

f2 – the divisor

ceil

```
public static double ceil(double a)
```

Returns the "ceiling" or smallest whole number greater than or equal to a.

Parameters:

a – an assigned value

floor

```
public static double floor(double a)
```

Returns the "floor" or largest whole number less than or equal to a.

Parameters:

a – an assigned value

rint

```
public static double rint(double a)
```

Converts a double value into an integral value in double format.

Parameters:

a – an assigned double value

atan2

```
public static double atan2(double a,
                           double b)
```

Converts rectangular coordinates (a, b) to polar (r, theta). This method computes the phase theta by computing an arc tangent of b/a in the range of –Pi to Pi.

Parameters:

a – an assigned value

b – an assigned value

Returns:

the polar coordinates (r, theta).

pow

```
public static double pow(double a,
                         double b) throws ArithmeticException
```

Returns the number a raised to the power of b. If (a == 0.0), then b must be greater than 0.0; otherwise you will throw an exception. An exception will also occur if (a (b > 0.0) && (a (b == a whole number)

b – an assigned value with the exceptions: (a == 0.0) –> (b > 0.0) && (a (b == a whole number)

Throws: ArithmeticException

If (a == 0.0) and (b **round**

```
public static int round(float a)
```

Rounds off a float value by first adding 0.5 to it and then returning the largest integer that is less than or equal to this new value.

Parameters:

a – the value to be rounded off

round

```
public static long round(double a)
```

Rounds off a double value by first adding 0.5 to it and then returning the largest integer that is less than or equal to this new value.

Parameters:

a – the value to be rounded off

random

```
public static synchronized double random()
```

Generates a random number between 0.0 and 1.0.

Random number generators are often referred to as pseudorandom number generators because the numbers produced tend to repeat themselves after a period of time.

Returns:

a pseudorandom double between 0.0 and 1.0.

abs

```
public static int abs(int a)
```

Returns the absolute integer value of a.

Parameters:

a – an assigned integer value

abs

```
public static long abs(long a)
```

Returns the absolute long value of a.

Parameters:

a – an assigned long value.

abs

```
public static float abs(float a)
```

Returns the absolute float value of a.

Parameters:

a – an assigned float value

abs

```
public static double abs(double a)
```

Returns the absolute double value of a.
Parameters:
a – an assigned double value

- **max**

```
public static int max(int a,
                      int b)
```

Takes two int values, a and b, and returns the greater number of the two.
Parameters:
a – an integer value to be compared
b – an integer value to be compared

- **max**

```
public static long max(long a,
                       long b)
```

Takes two long values, a and b, and returns the greater number of the two.
Parameters:
a – a long value to be compared
b – a long value to be compared

- **max**

```
public static float max(float a,
                        float b)
```

Takes two float values, a and b, and returns the greater number of the two.
Parameters:
a – a float value to be compared
b – a float value to be compared

- **max**

```
public static double max(double a,
                         double b)
```

Takes two double values, a and b, and returns the greater number of the two.
Parameters:
a – a double value to be compared
b – a double value to be compared

- **min**

```
public static int min(int a,
                      int b)
```

Takes two integer values, a and b, and returns the smallest number of the two.

Parameters:

a – an integer value to be compared
b – an integer value to be compared

min

```
public static long min(long a,
                       long b)
```

Takes two long values, a and b, and returns the smallest number of the two.

Parameters:

a – a long value to be compared
b – a long value to be compared

min

```
public static float min(float a,
                        float b)
```

Takes two float values, a and b, and returns the smallest number of the two.

Parameters:

a – a float value to be compared
b – a float value to be compared

min

```
public static double min(double a,
                         double b)
```

Takes two double values, a and b, and returns the smallest number of the two.

Parameters:

a – a double value to be compared
b – a double value to be compared

Class java.lang.NegativeArraySizeException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.NegativeArraySizeException
```

---

public class **NegativeArraySizeException**
extends RuntimeException
Signals that an attempt has been made to create an array with negative size.

Constructor Index

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

Constructors

NegativeArraySizeException

```
public NegativeArraySizeException()
```

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

NegativeArraySizeException

```
public NegativeArraySizeException(String s)
```

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

Parameters:

s – the detail message

Class java.lang.NoClassDefFoundError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
                           |
                           +----java.lang.NoClassDefFoundError
```

public class **NoClassDefFoundError**
extends LinkageError
Signals that a class could not be found.

Constructor Index

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

Constructors

NoClassDefFoundError

```
public NoClassDefFoundError()
```

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

NoClassDefFoundError

```
public NoClassDefFoundError(String s)
```

Constructs a NoClassDefFoundError with the specified detail message. A detail message is a String that describes this

particular exception.

Parameters:

s – the detail message

Class java.lang.NoSuchFieldError

```
java.lang.Object
    |
    +----java.lang.Throwable
             |
             +----java.lang.Error
                      |
                      +----java.lang.LinkageError
                               |
                               +----java.lang.IncompatibleClassChangeError
                                        |
                                        +----java.lang.NoSuchFieldError
```

---

public class **NoSuchFieldError**
extends IncompatibleClassChangeError
Signals that a particular field could not be found.

Constructor Index

- **NoSuchFieldError**()
  Constructs a NoSuchFieldException without a detail message.
- **NoSuchFieldError**(String)
  Constructs a NoSuchFieldException with a detail message.

Constructors

NoSuchFieldError

```
public NoSuchFieldError()
```

Constructs a NoSuchFieldException without a detail message. A detail message is a String that describes this particular exception.

NoSuchFieldError

```
public NoSuchFieldError(String s)
```

Constructs a NoSuchFieldException with a detail message. A detail message is a String that describes this particular

exception.

Parameters:

s – the detail message

Class java.lang.NoSuchMethodError

```
java.lang.Object
    |
    +----java.lang.Throwable
             |
             +----java.lang.Error
                      |
                      +----java.lang.LinkageError
                               |
                               +----java.lang.IncompatibleClassChangeError
                                        |
                                        +----java.lang.NoSuchMethodError
```

public class **NoSuchMethodError**
extends IncompatibleClassChangeError
Signals that a particular method could not be found.

Constructor Index

- **NoSuchMethodError**()
- **NoSuchMethodError**(String)
  Constructs a NoSuchMethodException with a detail message.

Constructors

NoSuchMethodError

```
public NoSuchMethodError()
```

NoSuchMethodError

```
public NoSuchMethodError(String s)
```

Constructs a NoSuchMethodException with a detail message. A detail message is a String that describes this particular exception.

Parameters:

s – the detail message

Class java.lang.NoSuchMethodException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.NoSuchMethodException
```

public class **NoSuchMethodException**
extends Exception
Signals that a particular method could not be found.

Constructor Index

- **NoSuchMethodException**()
  Constructs a NoSuchMethodException without a detail message.
- **NoSuchMethodException**(String)
  Constructs a NoSuchMethodException with a detail message.

Constructors

NoSuchMethodException

```
public NoSuchMethodException()
```

Constructs a NoSuchMethodException without a detail message. A detail message is a String that describes this particular exception.

NoSuchMethodException

```
public NoSuchMethodException(String s)
```

Constructs a NoSuchMethodException with a detail message. A detail message is a String that describes this particular exception.
Parameters:

s – the detail message

Class java.lang.NullPointerException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.NullPointerException
```

---

public class **NullPointerException**
extends RuntimeException
Signals the illegal use of a null pointer.

Constructor Index

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

Constructors

NullPointerException

```
public NullPointerException()
```

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

NullPointerException

```
public NullPointerException(String s)
```

Constructs a NullPointerException with the specified detail message. A detail message is a String that describes this

particular exception.

Parameters:

s – the detail message

Class java.lang.Number

```
java.lang.Object
    |
    +----java.lang.Number
```

---

public class **Number**
extends Object
Number is an abstract superclass for numeric scalar types. Integer, Long, Float and Double are subclasses of Number that bind to a particular numeric representation.

See Also:
Integer, Long, Float, Double

Constructor Index

- **Number**()

Method Index

- **doubleValue**()
  Returns the value of the number as a double.
- **floatValue**()
  Returns the value of the number as a float.
- **intValue**()
  Returns the value of the number as an int.
- **longValue**()
  Returns the value of the number as a long.

Constructors

Number

```
public Number()
```

Methods

- **intValue**

```
public abstract int intValue()
```

  Returns the value of the number as an int. This may involve rounding if the number is not already an integer.

- **longValue**

```
public abstract long longValue()
```

  Returns the value of the number as a long. This may involve rounding if the number is not already a long.

- **floatValue**

```
public abstract float floatValue()
```

  Returns the value of the number as a float. This may involve rounding if the number is not already a float.

- **doubleValue**

```
public abstract double doubleValue()
```

  Returns the value of the number as a double. This may involve rounding if the number is not already a double.

Class java.lang.NumberFormatException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.IllegalArgumentException
                                   |
                                   +----java.lang.NumberFormatException
```

---

public class **NumberFormatException**
extends IllegalArgumentException
Signals that an invalid number format has occurred.
See Also:
toString

Constructor Index

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

Constructors

NumberFormatException

```
public NumberFormatException()
```

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

NumberFormatException

```
public NumberFormatException(String s)
```

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

Parameters:

s – the detail message

Class java.lang.Object

`java.lang.Object`

---

public class **Object**
The root of the Class hierarchy. Every Class in the system has Object as its ultimate parent. Every variable and method defined here is available in every Object.
See Also:
Class

Constructor Index

- **Object**()

Method Index

- **clone**()
  Creates a clone of the object.
- **equals**(Object)
  Compares two Objects for equality.
- **finalize**()
  Code to perform when this object is garbage collected.
- **getClass**()
  Returns the Class of this Object.
- **hashCode**()
  Returns a hashcode for this Object.
- **notify**()
  Notifies a single waiting thread on a change in condition of another thread.
- **notifyAll**()
  Notifies all of the threads waiting for a condition to change.
- **toString**()
  Returns a String that represents the value of this Object.
- **wait**(long)
  Causes a thread to wait until it is notified or the specified timeout expires.
- **wait**(long, int)
  More accurate wait.

- **wait**()
  Causes a thread to wait forever until it is notified.

Constructors

Object

```
public Object()
```

Methods

getClass

```
public final Class getClass()
```

Returns the Class of this Object. Java has a runtime representation for classes– a descriptor of type Class– which the method getClass() returns for any Object.

hashCode

```
public int hashCode()
```

Returns a hashcode for this Object. Each Object in the Java system has a hashcode. The hashcode is a number that is usually different for different Objects. It is used when storing Objects in hashtables. Note: hashcodes can be negative as well as positive.

See Also:

Hashtable

equals

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

Compares two Objects for equality. Returns a boolean that indicates whether this Object is equivalent to the specified Object. This method is used when an Object is stored in a hashtable.

Parameters:

obj – the Object to compare with

Returns:

true if these Objects are equal; false otherwise.

See Also:

Hashtable

clone

```
protected Object clone() throws CloneNotSupportedException
```

Creates a clone of the object. A new instance is allocated and a bitwise clone of the current object is place in the new object.

Returns:

a clone of this Object.

Throws: OutOfMemoryError

If there is not enough memory.

Throws: CloneNotSupportedException

Object explicitly does not want to be cloned, or it does not support the Cloneable interface.

- **toString**

```
public String toString()
```

Returns a String that represents the value of this Object. It is recommended that all subclasses override this method.

- **notify**

```
public final void notify()
```

Notifies a single waiting thread on a change in condition of another thread. The thread effecting the change notifies the waiting thread using notify(). Threads that want to wait for a condition to change before proceeding can call wait().

The method notify() can only be called from within a synchronized method.

Throws: IllegalMonitorStateException

If the current thread is not the owner of the Object's monitor.

See Also:

wait, notifyAll

- **notifyAll**

```
public final void notifyAll()
```

Notifies all of the threads waiting for a condition to change. Threads that are waiting are generally waiting for another thread to change some condition. Thus, the thread effecting a change that more than one thread is waiting for notifies all the waiting threads using the method notifyAll(). Threads that want to wait for a condition to change before proceeding can call wait().

The method notifyAll() can only be called from within a synchronized method.

Throws: IllegalMonitorStateException

If the current thread is not the owner of the Object's

monitor.

See Also:

wait, notify

wait

```
public final void wait(long timeout) throws InterruptedException
```

Causes a thread to wait until it is notified or the specified timeout expires.

The method wait() can only be called from within a synchronized method.

Parameters:

timeout – the maximum time to wait in milliseconds

Throws: IllegalMonitorStateException

If the current thread is not the owner of the Object's monitor.

Throws: InterruptedException

Another thread has interrupted this thread.

wait

```
public final void wait(long timeout,
                       int nanos) throws InterruptedException
```

More accurate wait. *The method wait() can only be called from within a synchronized method.*

Parameters:

timeout – the maximum time to wait in milliseconds

nano – additional time, in nanoseconds range 0–999999

Throws: IllegalMonitorStateException

If the current thread is not the owner of the Object's monitor.

Throws: InterruptedException

Another thread has interrupted this thread.

wait

```
public final void wait() throws InterruptedException
```

Causes a thread to wait forever until it is notified.

The method wait() can only be called from within a synchronized method

Throws: IllegalMonitorStateException

If the current thread is not the owner of the Object's monitor.

Throws: InterruptedException

Another thread has interrupted this thread.

finalize

```
protected void finalize() throws Throwable
```

Code to perform when this object is garbage collected. The default is that nothing needs to be performed. Any exception thrown by a finalize method causes the finalization to halt. But otherwise, it is ignored.

Class java.lang.OutOfMemoryError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.VirtualMachineError
                           |
                           +----java.lang.OutOfMemoryError
```

---

public class **OutOfMemoryError**
extends VirtualMachineError
Signals that you are out of memory.

Constructor Index

- **OutOfMemoryError**()
  Constructs an OutOfMemoryError with no detail message.
- **OutOfMemoryError**(String)
  Constructs an OutOfMemoryError with the specified detail message.

Constructors

OutOfMemoryError

```
public OutOfMemoryError()
```

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

OutOfMemoryError

```
public OutOfMemoryError(String s)
```

Constructs an OutOfMemoryError with the specified detail message. A detail message is a String that describes this particular exception.
Parameters:

s – the detail message

Class java.lang.Process

```
java.lang.Object
    |
    +----java.lang.Process
```

---

public class **Process**
extends Object
An instance of class Process is returned by variants of the exec method in class System. From the Process instance, it is possible to: get the standin and/or standout of the subprocess, kill the subprocess, wait for it to terminate, and to retrieve the final exit value of the process.

Dropping the last reference to a Process instance does *not* kill the subprocess. There is no requirement that the subprocess execute asynchronously with the existing Java process.

Constructor Index

- **Process**()

Method Index

- **destroy**()
  Kills the subprocess.
- **exitValue**()
  Returns the exit value for the subprocess.
- **getErrorStream**()
  Returns the an InputStream connected to the error stream of the child process.
- **getInputStream**()
  Returns a Stream connected to the output of the child process.
- **getOutputStream**()
  Returns a Stream connected to the input of the child process.
- **waitFor**()
  Waits for the subprocess to complete.

Constructors

- **Process**

```
public Process()
```

Methods

- **getOutputStream**

```
public abstract OutputStream getOutputStream()
```

  Returns a Stream connected to the input of the child process. This stream is traditionally buffered.

- **getInputStream**

```
public abstract InputStream getInputStream()
```

  Returns a Stream connected to the output of the child process. This stream is traditionally buffered.

- **getErrorStream**

```
public abstract InputStream getErrorStream()
```

  Returns the an InputStream connected to the error stream of the child process. This stream is traditionally unbuffered.

- **waitFor**

```
public abstract int waitFor() throws InterruptedException
```

  Waits for the subprocess to complete. If the subprocess has already terminated the exit value is simply returned. If the subprocess has not yet terminated the calling thread will be blocked until the subprocess exits. Should this be called join()?

  Throws: InterruptedException

  Another thread has interrupted this thread.

- **exitValue**

```
public abstract int exitValue()
```

  Returns the exit value for the subprocess.

  Throws: IllegalThreadStateException

  If the subprocess has not yet terminated.

- **destroy**

```
public abstract void destroy()
```

Kills the subprocess.

Interface java.lang.Runnable

public interface **Runnable**
extends Object
This interface is designed to provide a common protocol for Objects that wish to execute code while they are active. For example, Runnable is implemented by class Thread. Being active simply means that a thread has been started and has not yet been stopped.

In addition, Runnable provides the means for a class to be active while not subclassing Thread. A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target. In most cases, the Runnable interface should be used if you are only planning to override the run() method and no other Thread methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.
See Also:
Thread

- **run**()
  The method that is executed when a Runnable object is activated.

- **run**

```
public abstract void run()
```

The method that is executed when a Runnable object is activated. The run() method is the "soul" of a Thread. It is in this method that all of the action of a Thread takes place.
See Also:
run

Class java.lang.Runtime

```
java.lang.Object
    |
    +----java.lang.Runtime
```

---

public class **Runtime**
extends Object

Method Index

- **exec**(String)
  Executes the system command specified in the parameter.
- **exec**(String, String[])
  Executes the system command specified in the parameter.
- **exec**(String[])
  Executes the system command specified by cmdarray[0] with arguments specified by the strings in the rest of the array.
- **exec**(String[], String[])
  Executes the system command specified by cmdarray[0] with arguments specified by the strings in the rest of the array.
- **exit**(int)
  Exits the virtual machine with an exit code.
- **freeMemory**()
  Returns the number of free bytes in system memory.
- **gc**()
  Runs the garbage collector.
- **getLocalizedInputStream**(InputStream)
  Localize an input stream.
- **getLocalizedOutputStream**(OutputStream)
  Localize an output stream.
- **getRuntime**()
  Return the runtime.
- **load**(String)
  Loads a dynamic library, given a complete path name.
- **loadLibrary**(String)
  Loads a dynamic library with the specified library name.
- **runFinalization**()
  Runs the finalization methods of any objects pending finalization.
- **totalMemory**()
  Returns the total number of bytes in system memory.

- **traceInstructions**(boolean)
  Enables/Disables tracing of instructions.
- **traceMethodCalls**(boolean)
  Enables/Disables tracing of method calls.

getRuntime

```
public static Runtime getRuntime()
```

Return the runtime.

exit

```
public void exit(int status)
```

Exits the virtual machine with an exit code. This method does not return, use with caution.

Parameters:

status – exit status, 0 if successful, other values indicate various error types.

exec

```
public Process exec(String command) throws IOException
```

Executes the system command specified in the parameter. Returns a Process which has methods for optaining the stdin, stdout, and stderr of the subprocess. This method fails if executed by untrusted code.

Parameters:

command – a specified system command

Returns:

an instance of class Process

exec

```
public Process exec(String command,
                    String envp[]) throws IOException
```

Executes the system command specified in the parameter. Returns a Process which has methods for optaining the stdin, stdout, and stderr of the subprocess. This method fails if executed by untrusted code.

Parameters:

command – a specified system command

Returns:

an instance of class Process

exec

```
public Process exec(String cmdarray[]) throws IOException
```

Executes the system command specified by cmdarray[0] with arguments specified by the strings in the rest of the array. Returns a Process which has methods for optaining the stdin, stdout, and stderr of the subprocess. This method fails if executed by untrusted code.

Parameters:

an – array containing the command to call and its arguments
envp – array containing environment in format name=value

Returns:

an instance of class Process

- **exec**

```
public Process exec(String cmdarray[],
                    String envp[]) throws IOException
```

Executes the system command specified by cmdarray[0] with arguments specified by the strings in the rest of the array. Returns a Process which has methods for optaining the stdin, stdout, and stderr of the subprocess. This method fails if executed by untrusted code.

Parameters:

an – array containing the command to call and its arguments
envp – array containing environment in format name=value

Returns:

an instance of class Process

- **freeMemory**

```
public long freeMemory()
```

Returns the number of free bytes in system memory. This number is not always accurate because it is just an estimation of the available memory. More memory may be freed by calling System.gc() .

- **totalMemory**

```
public long totalMemory()
```

Returns the total number of bytes in system memory.

- **gc**

```
public void gc()
```

Runs the garbage collector.

- **runFinalization**

```
public void runFinalization()
```

  Runs the finalization methods of any objects pending finalization. Usually you will not need to call this method since finalization methods will be called asynchronously by the finalization thread. However, under some circumstances (like running out of a finalized resource) it can be useful to run finalization methods synchronously.

- **traceInstructions**

```
public void traceInstructions(boolean on)
```

  Enables/Disables tracing of instructions.
  Parameters:
    on – start tracing if true

- **traceMethodCalls**

```
public void traceMethodCalls(boolean on)
```

  Enables/Disables tracing of method calls.
  Parameters:
    on – start tracing if true

- **load**

```
public synchronized void load(String filename)
```

  Loads a dynamic library, given a complete path name. If you use this from java_g it will automagically insert "_g" before the ".so". Example:
  Runtime.getRuntime().load("/home/avh/lib/libX11.so");
  Parameters:
    filename – the file to load
  Throws: UnsatisfiedLinkError
    If the file does not exist.
  See Also:
    getRuntime

- **loadLibrary**

```
public synchronized void loadLibrary(String libname)
```

  Loads a dynamic library with the specified library name. The call to LoadLibrary() should be made in the static initializer of the first class that is loaded. Linking in the same library more than once is ignored.
  Parameters:
    libname – the name of the library
  Throws: UnsatisfiedLinkError

If the library does not exist.

- **getLocalizedInputStream**

```
public InputStream getLocalizedInputStream(InputStream in)
```

Localize an input stream. A localized input stream will automatically translate the input from the local format to UNICODE.

- **getLocalizedOutputStream**

```
public OutputStream getLocalizedOutputStream(OutputStream out)
```

Localize an output stream. A localized output stream will automatically translate the output from UNICODE to the local format.

Class java.lang.RuntimeException

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

---

public class **RuntimeException**
extends Exception
An exception that can reasonably occur during the execution of a Java program by the Virtual machine.

Constructor Index

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

Constructors

RuntimeException

```
public RuntimeException()
```

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

RuntimeException

```
public RuntimeException(String s)
```

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

Parameters:

s – the detail message

Class java.lang.SecurityException

```
java.lang.Object
    |
    +----java.lang.Throwable
              |
              +----java.lang.Exception
                        |
                        +----java.lang.RuntimeException
                                  |
                                  +----java.lang.SecurityException
```

---

public class **SecurityException**
extends RuntimeException
Signals that a security exception has occurred.

Constructor Index

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

Constructors

SecurityException

```
public SecurityException()
```

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

SecurityException

```
public SecurityException(String s)
```

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

s – the detail message

Class java.lang.SecurityManager

```
java.lang.Object
    |
    +----java.lang.SecurityManager
```

---

public class **SecurityManager**
extends Object
Security Manager. An abstract class that can be subclassed to implement a security policy. It allows the inspection of the classloaders on the execution stack.

Variable Index

- **inCheck**

Constructor Index

- **SecurityManager**()
  Constructs a new SecurityManager.

Method Index

- **checkAccept**(String, int)
  Checks to see if a socket connection to the specified port on the specified host has been accepted.
- **checkAccess**(Thread)
  Checks to see if the specified Thread is allowed to modify the Thread group.
- **checkAccess**(ThreadGroup)
  Checks to see if the specified Thread group is allowed to modify this group.
- **checkConnect**(String, int)
  Checks to see if a socket has connected to the specified port on the the specified host.
- **checkConnect**(String, int, Object)
  Checks to see if the current execution context and the indicated execution context are both allowed to connect to the

indicated host and port.

- **checkCreateClassLoader**()
  Checks to see if the ClassLoader has been created.
- **checkDelete**(String)
  Checks to see if a file with the specified system dependent file name can be deleted.
- **checkExec**(String)
  Checks to see if the system command is executed by trusted code.
- **checkExit**(int)
  Checks to see if the system has exited the virtual machine with an exit code.
- **checkLink**(String)
  Checks to see if the specified linked library exists.
- **checkListen**(int)
  Checks to see if a server socket is listening to the specified local port that it is bounded to.
- **checkPackageAccess**(String)
  Check if an applet can access a package.
- **checkPackageDefinition**(String)
  Check if an applet can define classes in a package.
- **checkPropertiesAccess**()
  Checks to see who has access to the System properties.
- **checkPropertyAccess**(String)
  Checks to see who has access to the System property named by *key*
- **checkPropertyAccess**(String, String)
  Checks to see who has access to the System property named by *key* and *def*
- **checkRead**(FileDescriptor)
  Checks to see if an input file with the specified file descriptor object gets created.
- **checkRead**(String)
  Checks to see if an input file with the specified system dependent file name gets created.
- **checkRead**(String, Object)
  Checks to see if the current context or the indicated context are both allowed to read the given file name.
- **checkSetFactory**()
  Check if an applet can set a networking−related object factory.
- **checkTopLevelWindow**(Object)
  Checks to see if top−level windows can be created by the caller.
- **checkWrite**(FileDescriptor)
  Checks to see if an output file with the specified file descriptor object gets created.
- **checkWrite**(String)
  Checks to see if an output file with the specified system

dependent file name gets created.

- **classDepth**(String)
  Return the position of the stack frame containing the first occurrence of the named class.
- **classLoaderDepth**()
- **currentClassLoader**()
  The current ClassLoader on the execution stack.
- **getClassContext**()
  Gets the context of this Class.
- **getInCheck**()
  Returns whether there is a security check in progress.
- **getSecurityContext**()
  Returns an implementation−dependent Object which encapsulates enough information about the current execution environment to perform some of the security checks later.
- **inClass**(String)
  Returns true if the specified String is in this Class.
- **inClassLoader**()
  Returns a boolean indicating whether or not the current ClassLoader is equal to null.

Variables

inCheck

```
protected boolean inCheck
```

Constructors

SecurityManager

```
protected SecurityManager()
```

Constructs a new SecurityManager.
Throws: SecurityException
If the security manager cannot be created.

Methods

getInCheck

```
public boolean getInCheck()
```

Returns whether there is a security check in progress.

- **getClassContext**

```
protected Class[] getClassContext()
```

Gets the context of this Class.

- **currentClassLoader**

```
protected ClassLoader currentClassLoader()
```

The current ClassLoader on the execution stack.

- **classDepth**

```
protected int classDepth(String name)
```

Return the position of the stack frame containing the first occurrence of the named class.

Parameters:

name – classname of the class to search for

- **classLoaderDepth**

```
protected int classLoaderDepth()
```

- **inClass**

```
protected boolean inClass(String name)
```

Returns true if the specified String is in this Class.

Parameters:

name – the name of the class

- **inClassLoader**

```
protected boolean inClassLoader()
```

Returns a boolean indicating whether or not the current ClassLoader is equal to null.

- **getSecurityContext**

```
public Object getSecurityContext()
```

Returns an implementation–dependent Object which encapsulates enough information about the current execution environment to perform some of the security checks later.

- **checkCreateClassLoader**

```
public void checkCreateClassLoader()
```

Checks to see if the ClassLoader has been created.

Throws: SecurityException

If a security error has occurred.

- **checkAccess**

```
public void checkAccess(Thread g)
```

Checks to see if the specified Thread is allowed to modify the Thread group.

Parameters:

g – the Thread to be checked

Throws: SecurityException

If the current Thread is not allowed to access this Thread group.

- **checkAccess**

```
public void checkAccess(ThreadGroup g)
```

Checks to see if the specified Thread group is allowed to modify this group.

Parameters:

g – the Thread group to be checked

Throws: SecurityException

If the current Thread group is not allowed to access this Thread group.

- **checkExit**

```
public void checkExit(int status)
```

Checks to see if the system has exited the virtual machine with an exit code.

Parameters:

status – exit status, 0 if successful, other values indicate various error types.

Throws: SecurityException

If a security error has occurred.

- **checkExec**

```
public void checkExec(String cmd)
```

Checks to see if the system command is executed by trusted code.

Parameters:

cmd – the specified system command

Throws: SecurityException

If a security error has occurred.

- **checkLink**

```
public void checkLink(String lib)
```

Checks to see if the specified linked library exists.

Parameters:
lib – the name of the library
Throws: SecurityException
If the library does not exist.

- **checkRead**

```
public void checkRead(FileDescriptor fd)
```

Checks to see if an input file with the specified file descriptor object gets created.
Parameters:
fd – the system dependent file descriptor
Throws: SecurityException
If a security error has occurred.

- **checkRead**

```
public void checkRead(String file)
```

Checks to see if an input file with the specified system dependent file name gets created.
Parameters:
file – the system dependent file name
Throws: SecurityException
If the file is not found.

- **checkRead**

```
public void checkRead(String file,
                      Object context)
```

Checks to see if the current context or the indicated context are both allowed to read the given file name.
Parameters:
file – the system dependent file name
context – the alternate execution context which must also be checked
Throws: SecurityException
If the file is not found.

- **checkWrite**

```
public void checkWrite(FileDescriptor fd)
```

Checks to see if an output file with the specified file descriptor object gets created.
Parameters:
fd – the system dependent file descriptor
Throws: SecurityException
If a security error has occurred.

- **checkWrite**

```
public void checkWrite(String file)
```

Checks to see if an output file with the specified system dependent file name gets created.

Parameters:

file – the system dependent file name

Throws: SecurityException

If the file is not found.

- **checkDelete**

```
public void checkDelete(String file)
```

Checks to see if a file with the specified system dependent file name can be deleted.

Parameters:

file – the system dependent file name

Throws: SecurityException

If the file is not found.

- **checkConnect**

```
public void checkConnect(String host,
                         int port)
```

Checks to see if a socket has connected to the specified port on the the specified host.

Parameters:

host – the host name port to connect to

port – the protocol port to connect to

Throws: SecurityException

If a security error has occurred.

- **checkConnect**

```
public void checkConnect(String host,
                         int port,
                         Object context)
```

Checks to see if the current execution context and the indicated execution context are both allowed to connect to the indicated host and port.

- **checkListen**

```
public void checkListen(int port)
```

Checks to see if a server socket is listening to the specified local port that it is bounded to.

Parameters:

port – the protocol port to connect to

Throws: SecurityException

If a security error has occurred.

- **checkAccept**

```
public void checkAccept(String host,
                        int port)
```

Checks to see if a socket connection to the specified port on the specified host has been accepted.

Parameters:

host – the host name to connect to

port – the protocol port to connect to

Throws: SecurityException

If a security error has occurred.

- **checkPropertiesAccess**

```
public void checkPropertiesAccess()
```

Checks to see who has access to the System properties.

Throws: SecurityException

If a security error has occurred.

- **checkPropertyAccess**

```
public void checkPropertyAccess(String key)
```

Checks to see who has access to the System property named by *key*

Parameters:

key – the System property that the caller wants to examine

Throws: SecurityException

If a security error has occurred.

- **checkPropertyAccess**

```
public void checkPropertyAccess(String key,
                                String def)
```

Checks to see who has access to the System property named by *key* and *def*

Parameters:

key – the System property that the caller wants to examine

def – default value to return if this property is not defined

Throws: SecurityException

If a security error has occurred.

- **checkTopLevelWindow**

```
public boolean checkTopLevelWindow(Object window)
```

Checks to see if top–level windows can be created by the caller. A return of false means that the window creation is allowed but the window should indicate some sort of visual warning. Returning true means the creation is allowed with

no special restrictions. To disallow the creation entirely, this method should throw a SecurityException.

Parameters:

window – the new window that's being created.

- **checkPackageAccess**

```
public void checkPackageAccess(String pkg)
```

Check if an applet can access a package.

- **checkPackageDefinition**

```
public void checkPackageDefinition(String pkg)
```

Check if an applet can define classes in a package.

- **checkSetFactory**

```
public void checkSetFactory()
```

Check if an applet can set a networking–related object factory.

Class java.lang.StackOverflowError

```
java.lang.Object
    |
    +----java.lang.Throwable
              |
              +----java.lang.Error
                        |
                        +----java.lang.VirtualMachineError
                                  |
                                  +----java.lang.StackOverflowError
```

---

public class **StackOverflowError**
extends VirtualMachineError
Signals that a stack overflow has occurred.

Constructor Index

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

Constructors

StackOverflowError

```
public StackOverflowError()
```

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

StackOverflowError

```
public StackOverflowError(String s)
```

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

s – the detail message

Class java.lang.String

```
java.lang.Object
    |
    +----java.lang.String
```

---

public final class **String**
extends Object
A general class of objects to represent character Strings. Strings are constant, their values cannot be changed after creation. The compiler makes sure that each String constant actually results in a String object. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

See Also:
StringBuffer

Constructor Index

- **String**()
  Constructs a new empty String.
- **String**(String)
  Constructs a new String that is a copy of the specified String.
- **String**(char[])
  Constructs a new String whose initial value is the specified array of characters.
- **String**(char[], int, int)
  Constructs a new String whose initial value is the specified

sub array of characters.

- **String**(byte[], int, int, int)
  Constructs a new String whose initial value is the specified sub array of bytes.
- **String**(byte[], int)
  Constructs a new String whose value is the specified array of bytes.
- **String**(StringBuffer)
  Construct a new string whose value is the current contents of the given string buffer

Method Index

- **charAt**(int)
  Returns the character at the specified index.
- **compareTo**(String)
  Compares this String to another specified String.
- **concat**(String)
  Concatenates the specified string to the end of this String.
- **copyValueOf**(char[], int, int)
  Returns a String that is equivalent to the specified character array.
- **copyValueOf**(char[])
  Returns a String that is equivalent to the specified character array.
- **endsWith**(String)
  Determines whether the String ends with some suffix.
- **equals**(Object)
  Compares this String to the specified object.
- **equalsIgnoreCase**(String)
  Compares this String to another object.
- **getBytes**(int, int, byte[], int)
  Copies characters from this String into the specified byte array.
- **getChars**(int, int, char[], int)
  Copies characters from this String into the specified character array.
- **hashCode**()
  Returns a hashcode for this String.
- **indexOf**(int)
  Returns the index within this String of the first occurrence of the specified character.
- **indexOf**(int, int)
  Returns the index within this String of the first occurrence of the specified character, starting the search at fromIndex.
- **indexOf**(String)
  Returns the index within this String of the first occurrence of

the specified substring.

- **indexOf**(String, int)
  Returns the index within this String of the first occurrence of the specified substring.
- **intern**()
  Returns a String that is equal to this String but which is guaranteed to be from the unique String pool.
- **lastIndexOf**(int)
  Returns the index within this String of the last occurrence of the specified character.
- **lastIndexOf**(int, int)
  Returns the index within this String of the last occurrence of the specified character.
- **lastIndexOf**(String)
  Returns the index within this String of the last occurrence of the specified substring.
- **lastIndexOf**(String, int)
  Returns the index within this String of the last occurrence of the specified substring.
- **length**()
  Returns the length of the String.
- **regionMatches**(int, String, int, int)
  Determines whether a region of this String matches the specified region of the specified String.
- **regionMatches**(boolean, int, String, int, int)
  Determines whether a region of this String matches the specified region of the specified String.
- **replace**(char, char)
  Converts this String by replacing all occurences of oldChar with newChar.
- **startsWith**(String, int)
  Determines whether this String starts with some prefix.
- **startsWith**(String)
  Determines whether this String starts with some prefix.
- **substring**(int)
  Returns the substring of this String.
- **substring**(int, int)
  Returns the substring of a String.
- **toCharArray**()
  Converts this String to a character array.
- **toLowerCase**()
  Converts all of the characters in this String to lower case.
- **toString**()
  Converts this String to a String.
- **toUpperCase**()
  Converts all of the characters in this String to upper case.
- **trim**()
  Trims leading and trailing whitespace from this String.
- **valueOf**(Object)

Returns a String that represents the String value of the object.

- **valueOf**(char[])
  Returns a String that is equivalent to the specified character array.
- **valueOf**(char[], int, int)
  Returns a String that is equivalent to the specified character array.
- **valueOf**(boolean)
  Returns a String object that represents the state of the specified boolean.
- **valueOf**(char)
  Returns a String object that contains a single character
- **valueOf**(int)
  Returns a String object that represents the value of the specified integer.
- **valueOf**(long)
  Returns a String object that represents the value of the specified long.
- **valueOf**(float)
  Returns a String object that represents the value of the specified float.
- **valueOf**(double)
  Returns a String object that represents the value of the specified double.

Constructors

String

```
public String()
```

Constructs a new empty String.

String

```
public String(String value)
```

Constructs a new String that is a copy of the specified String.

Parameters:

value – the initial value of the String

String

```
public String(char value[])
```

Constructs a new String whose initial value is the specified array of characters.

Parameters:

value – the initial value of the String

String

```
public String(char value[],
              int offset,
              int count)
```

Constructs a new String whose initial value is the specified sub array of characters. The length of the new string will be count characters starting at offset within the specified character array.

Parameters:

value – the initial value of the String, an array of characters

offset – the offset into the value of the String

count – the length of the value of the String

Throws: StringIndexOutOfBoundsException

If the offset and count arguments are invalid.

String

```
public String(byte ascii[],
              int hibyte,
              int offset,
              int count)
```

Constructs a new String whose initial value is the specified sub array of bytes. The high–byte of each character can be specified, it should usually be 0. The length of the new String will be count characters starting at offset within the specified character array.

Parameters:

ascii – the bytes that will be converted to characters

hibyte – the high byte of each Unicode character

offset – the offset into the ascii array

count – the length of the String

Throws: StringIndexOutOfBoundsException

If the offset and count arguments are invalid.

String

```
public String(byte ascii[],
              int hibyte)
```

Constructs a new String whose value is the specified array of bytes. The byte array transformed into Unicode chars using hibyte as the upper byte of each character.

Parameters:

ascii – the byte that will be converted to characters

hibyte – the top 8 bits of each 16 bit Unicode character

String

```
public String(StringBuffer buffer)
```

Construct a new string whose value is the current contents of the given string buffer

Parameters:

buffer – the stringbuffer to be converted

- **length**

```
public int length()
```

Returns the length of the String. The length of the String is equal to the number of 16 bit Unicode characters in the String.

- **charAt**

```
public char charAt(int index)
```

Returns the character at the specified index. An index ranges from `0` to `length() - 1`.

Parameters:

index – the index of the desired character

Throws: StringIndexOutOfBoundsException

If the index is not in the range `0` to `length()-1`.

- **getChars**

```
public void getChars(int srcBegin,
                     int srcEnd,
                     char dst[],
                     int dstBegin)
```

Copies characters from this String into the specified character array. The characters of the specified substring (determined by srcBegin and srcEnd) are copied into the character array, starting at the array's dstBegin location.

Parameters:

srcBegin – index of the first character in the string
srcEnd – end of the characters that are copied
dst – the destination array
dstBegin – the start offset in the destination array

- **getBytes**

```
public void getBytes(int srcBegin,
                     int srcEnd,
                     byte dst[],
                     int dstBegin)
```

Copies characters from this String into the specified byte

array. Copies the characters of the specified substring (determined by srcBegin and srcEnd) into the byte array, starting at the array's dstBegin location.

Parameters:

srcBegin – index of the first character in the String
srcEnd – end of the characters that are copied
dst – the destination array
dstBegin – the start offset in the destination array

- **equals**

```
public boolean equals(Object anObject)
```

Compares this String to the specified object. Returns true if the object is equal to this String; that is, has the same length and the same characters in the same sequence.

Parameters:

anObject – the object to compare this String against

Returns:

true if the Strings are equal; false otherwise.

Overrides:

equals in class Object

- **equalsIgnoreCase**

```
public boolean equalsIgnoreCase(String anotherString)
```

Compares this String to another object. Returns true if the object is equal to this String; that is, has the same length and the same characters in the same sequence. Upper case characters are folded to lower case before they are compared.

Parameters:

anotherString – the String to compare this String against

Returns:

true if the Strings are equal, ignoring case; false otherwise.

- **compareTo**

```
public int compareTo(String anotherString)
```

Compares this String to another specified String. Returns an integer that is less than, equal to, or greater than zero. The integer's value depends on whether this String is less than, equal to, or greater than anotherString.

Parameters:

anotherString – the String to be compared

- **regionMatches**

```
public boolean regionMatches(int toffset,
                             String other,
                             int ooffset,
```

```
                                int len)
```

Determines whether a region of this String matches the specified region of the specified String.

Parameters:

toffset – where to start looking in this String
other – the other String
ooffset – where to start looking in the other String
len – the number of characters to compare

Returns:

true if the region matches with the other; false otherwise.

- **regionMatches**

```
public boolean regionMatches(boolean ignoreCase,
                             int toffset,
                             String other,
                             int ooffset,
                             int len)
```

Determines whether a region of this String matches the specified region of the specified String. If the boolean ignoreCase is true, upper case characters are considered equivalent to lower case letters.

Parameters:

ignoreCase – if true, case is ignored
toffset – where to start looking in this String
other – the other String
ooffset – where to start looking in the other String
len – the number of characters to compare

Returns:

true if the region matches with the other; false otherwise.

- **startsWith**

```
public boolean startsWith(String prefix,
                          int toffset)
```

Determines whether this String starts with some prefix.

Parameters:

prefix – the prefix
toffset – where to begin looking in the the String

Returns:

true if the String starts with the specified prefix; false otherwise.

- **startsWith**

```
public boolean startsWith(String prefix)
```

Determines whether this String starts with some prefix.

Parameters:

prefix – the prefix

Returns:

true if the String starts with the specified prefix; false otherwise.

- **endsWith**

```
public boolean endsWith(String suffix)
```

Determines whether the String ends with some suffix.

Parameters:

suffix – the suffix

Returns:

true if the String ends with the specified suffix; false otherwise.

- **hashCode**

```
public int hashCode()
```

Returns a hashcode for this String. This is a large number composed of the character values in the String.

Overrides:

hashCode in class Object

- **indexOf**

```
public int indexOf(int ch)
```

Returns the index within this String of the first occurrence of the specified character. This method returns −1 if the index is not found.

Parameters:

ch – the character to search for

- **indexOf**

```
public int indexOf(int ch,
                   int fromIndex)
```

Returns the index within this String of the first occurrence of the specified character, starting the search at fromIndex. This method returns −1 if the index is not found.

Parameters:

ch – the character to search for

fromIndex – the index to start the search from

- **lastIndexOf**

```
public int lastIndexOf(int ch)
```

Returns the index within this String of the last occurrence of the specified character. The String is searched backwards

starting at the last character. This method returns −1 if the index is not found.

Parameters:

ch − the character to search for

- **lastIndexOf**

```
public int lastIndexOf(int ch,
                       int fromIndex)
```

Returns the index within this String of the last occurrence of the specified character. The String is searched backwards starting at fromIndex. This method returns −1 if the index is not found.

Parameters:

ch − the character to search for
fromIndex − the index to start the search from

- **indexOf**

```
public int indexOf(String str)
```

Returns the index within this String of the first occurrence of the specified substring. This method returns −1 if the index is not found.

Parameters:

str − the substring to search for

- **indexOf**

```
public int indexOf(String str,
                   int fromIndex)
```

Returns the index within this String of the first occurrence of the specified substring. The search is started at fromIndex. This method returns −1 if the index is not found.

Parameters:

str − the substring to search for
fromIndex − the index to start the search from

- **lastIndexOf**

```
public int lastIndexOf(String str)
```

Returns the index within this String of the last occurrence of the specified substring. The String is searched backwards. This method returns −1 if the index is not found.

Parameters:

str − the substring to search for

- **lastIndexOf**

```
public int lastIndexOf(String str,
                       int fromIndex)
```

Returns the index within this String of the last occurrence of the specified substring. The String is searched backwards starting at fromIndex. This method returns −1 if the index is not found.

Parameters:

str – the substring to search for
fromIndex – the index to start the search from

- **substring**

```
public String substring(int beginIndex)
```

Returns the substring of this String. The substring is specified by a beginIndex (inclusive) and the end of the string.

Parameters:

beginIndex – the beginning index, inclusive

- **substring**

```
public String substring(int beginIndex,
                        int endIndex)
```

Returns the substring of a String. The substring is specified by a beginIndex (inclusive) and an endIndex (exclusive).

Parameters:

beginIndex – the beginning index, inclusive
endIndex – the ending index, exclusive

Throws: StringIndexOutOfBoundsException

If the beginIndex or the endIndex is out of range.

- **concat**

```
public String concat(String str)
```

Concatenates the specified string to the end of this String.

Parameters:

str – the String which is concatenated to the end of this String

- **replace**

```
public String replace(char oldChar,
                      char newChar)
```

Converts this String by replacing all occurences of oldChar with newChar.

Parameters:

oldChar – the old character
newChar – the new character

- **toLowerCase**

```
public String toLowerCase()
```

Converts all of the characters in this String to lower case.

Returns:
the String, converted to lowercase.
See Also:
toLowerCase, toUpperCase

toUpperCase

```
public String toUpperCase()
```

Converts all of the characters in this String to upper case.
Returns:
the String, converted to uppercase.
See Also:
toUpperCase, toLowerCase

trim

```
public String trim()
```

Trims leading and trailing whitespace from this String.
Returns:
the String, with whitespace removed.

toString

```
public String toString()
```

Converts this String to a String.
Returns:
the String itself.
Overrides:
toString in class Object

toCharArray

```
public char[] toCharArray()
```

Converts this String to a character array. This creates a new array.
Returns:
an array of characters.

valueOf

```
public static String valueOf(Object obj)
```

Returns a String that represents the String value of the object. The object may choose how to represent itself by implementing the toString() method.
Parameters:
obj – the object to be converted

valueOf

```
public static String valueOf(char data[])
```

Returns a String that is equivalent to the specified character array. Uses the original array as the body of the String (ie. it does not copy it to a new array).

Parameters:

data – the character array

- **valueOf**

```
public static String valueOf(char data[],
                             int offset,
                             int count)
```

Returns a String that is equivalent to the specified character array.

Parameters:

data – the character array
offset – the offset into the value of the String
count – the length of the value of the String

- **copyValueOf**

```
public static String copyValueOf(char data[],
                                 int offset,
                                 int count)
```

Returns a String that is equivalent to the specified character array. It creates a new array and copies the characters into it.

Parameters:

data – the character array
offset – the offset into the value of the String
count – the length of the value of the String

- **copyValueOf**

```
public static String copyValueOf(char data[])
```

Returns a String that is equivalent to the specified character array. It creates a new array and copies the characters into it.

Parameters:

data – the character array

- **valueOf**

```
public static String valueOf(boolean b)
```

Returns a String object that represents the state of the specified boolean.

Parameters:

b – the boolean

- **valueOf**

```
public static String valueOf(char c)
```

Returns a String object that contains a single character

Parameters:

c – the character

Returns:

the resulting String.

valueOf

```
public static String valueOf(int i)
```

Returns a String object that represents the value of the specified integer.

Parameters:

i – the integer

valueOf

```
public static String valueOf(long l)
```

Returns a String object that represents the value of the specified long.

Parameters:

l – the long

valueOf

```
public static String valueOf(float f)
```

Returns a String object that represents the value of the specified float.

Parameters:

f – the float

valueOf

```
public static String valueOf(double d)
```

Returns a String object that represents the value of the specified double.

Parameters:

d – the double

intern

```
public String intern()
```

Returns a String that is equal to this String but which is guaranteed to be from the unique String pool. For example:

`s1.intern() == s2.intern()  s1.equals(s2).`

Class java.lang.StringBuffer

```
java.lang.Object
    |
    +----java.lang.StringBuffer
```

---

public class **StringBuffer**
extends Object
This Class is a growable buffer for characters. It is mainly used to create Strings. The compiler uses it to implement the "+" operator. For example:

```
"a" + 4 + "c"
```

is compiled to:

```
new StringBuffer().append("a").append(4).append("c").toString()
```

Note that the method toString() does not create a copy of the internal buffer. Instead the buffer is marked as shared. Any further changes to the buffer will cause a copy to be made.

See Also:
String, ByteArrayOutputStream

Constructor Index

- **StringBuffer**()
  Constructs an empty String buffer.
- **StringBuffer**(int)
  Constructs an empty String buffer with the specified initial length.
- **StringBuffer**(String)
  Constructs a String buffer with the specified initial value.

Method Index

- **append**(Object)
  Appends an object to the end of this buffer.

- **append**(String)
  Appends a String to the end of this buffer.
- **append**(char[])
  Appends an array of characters to the end of this buffer.
- **append**(char[], int, int)
  Appends a part of an array of characters to the end of this buffer.
- **append**(boolean)
  Appends a boolean to the end of this buffer.
- **append**(char)
  Appends a character to the end of this buffer.
- **append**(int)
  Appends an integer to the end of this buffer.
- **append**(long)
  Appends a long to the end of this buffer.
- **append**(float)
  Appends a float to the end of this buffer.
- **append**(double)
  Appends a double to the end of this buffer.
- **capacity**()
  Returns the current capacity of the String buffer.
- **charAt**(int)
  Returns the character at the specified index.
- **copyWhenShared**()
  Copies the buffer value if it is shared.
- **ensureCapacity**(int)
  Ensures that the capacity of the buffer is at least equal to the specified minimum.
- **getChars**(int, int, char[], int)
  Copies the characters of the specified substring (determined by srcBegin and srcEnd) into the character array, starting at the array's dstBegin location.
- **insert**(int, Object)
  Inserts an object into the String buffer.
- **insert**(int, String)
  Inserts a String into the String buffer.
- **insert**(int, char[])
  Inserts an array of characters into the String buffer.
- **insert**(int, boolean)
  Inserts a boolean into the String buffer.
- **insert**(int, char)
  Inserts a character into the String buffer.
- **insert**(int, int)
  Inserts an integer into the String buffer.
- **insert**(int, long)
  Inserts a long into the String buffer.
- **insert**(int, float)
  Inserts a float into the String buffer.
- **insert**(int, double)

Inserts a double into the String buffer.

- **length**()
  Returns the length (character count) of the buffer.
- **setCharAt**(int, char)
  Changes the character at the specified index to be ch.
- **setLength**(int)
  Sets the length of the String.
- **toString**()
  Converts to a String representing the data in the buffer.

Constructors

StringBuffer

```
public StringBuffer()
```

Constructs an empty String buffer.

StringBuffer

```
public StringBuffer(int length)
```

Constructs an empty String buffer with the specified initial length.

Parameters:

length – the initial length

StringBuffer

```
public StringBuffer(String str)
```

Constructs a String buffer with the specified initial value.

Parameters:

str – the initial value of the buffer

Methods

length

```
public int length()
```

Returns the length (character count) of the buffer.

capacity

```
public int capacity()
```

Returns the current capacity of the String buffer. The

capacity is the amount of storage available for newly inserted characters; beyond which an allocation will occur.

- **copyWhenShared**

```
protected void copyWhenShared()
```

Copies the buffer value if it is shared.

- **ensureCapacity**

```
public synchronized void ensureCapacity(int minimumCapacity)
```

Ensures that the capacity of the buffer is at least equal to the specified minimum.

Parameters:

minimumCapacity – the minimum desired capacity

- **setLength**

```
public synchronized void setLength(int newLength)
```

Sets the length of the String. If the length is reduced, characters are lost. If the length is extended, the values of the new characters are set to 0.

Parameters:

newLength – the new length of the buffer

Throws: StringIndexOutOfBoundsException

If the length is invalid.

- **charAt**

```
public synchronized char charAt(int index)
```

Returns the character at the specified index. An index ranges from 0..length()–1.

Parameters:

index – the index of the desired character

Throws: StringIndexOutOfBoundsException

If the index is invalid.

- **getChars**

```
public synchronized void getChars(int srcBegin,
                                  int srcEnd,
                                  char dst[],
                                  int dstBegin)
```

Copies the characters of the specified substring (determined by srcBegin and srcEnd) into the character array, starting at the array's dstBegin location. Both srcBegin and srcEnd must be legal indexes into the buffer.

Parameters:

srcBegin – begin copy at this offset in the String

srcEnd – stop copying at this offset in the String

dst – the array to copy the data into
dstBegin – offset into dst

Throws: StringIndexOutOfBoundsException
If there is an invalid index into the buffer.

- **setCharAt**

```
public synchronized void setCharAt(int index,
                                   char ch)
```

Changes the character at the specified index to be ch.

Parameters:
index – the index of the character
ch – the new character

Throws: StringIndexOutOfBoundsException
If the index is invalid.

- **append**

```
public synchronized StringBuffer append(Object obj)
```

Appends an object to the end of this buffer.

Parameters:
obj – the object to be appended

Returns:
the StringBuffer itself, NOT a new one.

- **append**

```
public synchronized StringBuffer append(String str)
```

Appends a String to the end of this buffer.

Parameters:
str – the String to be appended

Returns:
the StringBuffer itself, NOT a new one.

- **append**

```
public synchronized StringBuffer append(char str[])
```

Appends an array of characters to the end of this buffer.

Parameters:
str – the characters to be appended

Returns:
the StringBuffer itself, NOT a new one.

- **append**

```
public synchronized StringBuffer append(char str[],
                                        int offset,
                                        int len)
```

Appends a part of an array of characters to the end of this buffer.

Parameters:
str – the characters to be appended
offset – where to start
len – the number of characters to add
Returns:
the StringBuffer itself, NOT a new one.

- **append**

```
public StringBuffer append(boolean b)
```

Appends a boolean to the end of this buffer.
Parameters:
b – the boolean to be appended
Returns:
the StringBuffer itself, NOT a new one.

- **append**

```
public synchronized StringBuffer append(char c)
```

Appends a character to the end of this buffer.
Parameters:
ch – the character to be appended
Returns:
the StringBuffer itself, NOT a new one.

- **append**

```
public StringBuffer append(int i)
```

Appends an integer to the end of this buffer.
Parameters:
i – the integer to be appended
Returns:
the StringBuffer itself, NOT a new one.

- **append**

```
public StringBuffer append(long l)
```

Appends a long to the end of this buffer.
Parameters:
l – the long to be appended
Returns:
the StringBuffer itself, NOT a new one.

- **append**

```
public StringBuffer append(float f)
```

Appends a float to the end of this buffer.
Parameters:
f – the float to be appended

Returns:
the StringBuffer itself, NOT a new one.

- **append**

```
public StringBuffer append(double d)
```

Appends a double to the end of this buffer.
Parameters:
d – the double to be appended
Returns:
the StringBuffer itself, NOT a new one.

- **insert**

```
public synchronized StringBuffer insert(int offset,
                                        Object obj)
```

Inserts an object into the String buffer.
Parameters:
offset – the offset at which to insert
obj – the object to insert
Returns:
the StringBuffer itself, NOT a new one.
Throws: StringIndexOutOfBoundsException
If the offset is invalid.

- **insert**

```
public synchronized StringBuffer insert(int offset,
                                        String str)
```

Inserts a String into the String buffer.
Parameters:
offset – the offset at which to insert
str – the String to insert
Returns:
the StringBuffer itself, NOT a new one.
Throws: StringIndexOutOfBoundsException
If the offset is invalid.

- **insert**

```
public synchronized StringBuffer insert(int offset,
                                        char str[])
```

Inserts an array of characters into the String buffer.
Parameters:
offset – the offset at which to insert
str – the characters to insert
Returns:
the StringBuffer itself, NOT a new one.
Throws: StringIndexOutOfBoundsException
If the offset is invalid.

- **insert**

```
public StringBuffer insert(int offset,
                           boolean b)
```

  Inserts a boolean into the String buffer.
  Parameters:
    offset – the offset at which to insert
    b – the boolean to insert
  Returns:
    the StringBuffer itself, NOT a new one.
  Throws: StringIndexOutOfBoundsException
    If the offset is invalid.

- **insert**

```
public synchronized StringBuffer insert(int offset,
                                        char c)
```

  Inserts a character into the String buffer.
  Parameters:
    offset – the offset at which to insert
    ch – the character to insert
  Returns:
    the StringBuffer itself, NOT a new one.
  Throws: StringIndexOutOfBoundsException
    If the offset invalid.

- **insert**

```
public StringBuffer insert(int offset,
                           int i)
```

  Inserts an integer into the String buffer.
  Parameters:
    offset – the offset at which to insert
    i – the integer to insert
  Returns:
    the StringBuffer itself, NOT a new one.
  Throws: StringIndexOutOfBoundsException
    If the offset is invalid.

- **insert**

```
public StringBuffer insert(int offset,
                           long l)
```

  Inserts a long into the String buffer.
  Parameters:
    offset – the offset at which to insert
    l – the long to insert
  Returns:
    the StringBuffer itself, NOT a new one.

Throws: StringIndexOutOfBoundsException
If the offset is invalid.

insert

```
public StringBuffer insert(int offset,
                           float f)
```

Inserts a float into the String buffer.
Parameters:
offset – the offset at which to insert
f – the float to insert
Returns:
the StringBuffer itself, NOT a new one.
Throws: StringIndexOutOfBoundsException
If the offset is invalid.

insert

```
public StringBuffer insert(int offset,
                           double d)
```

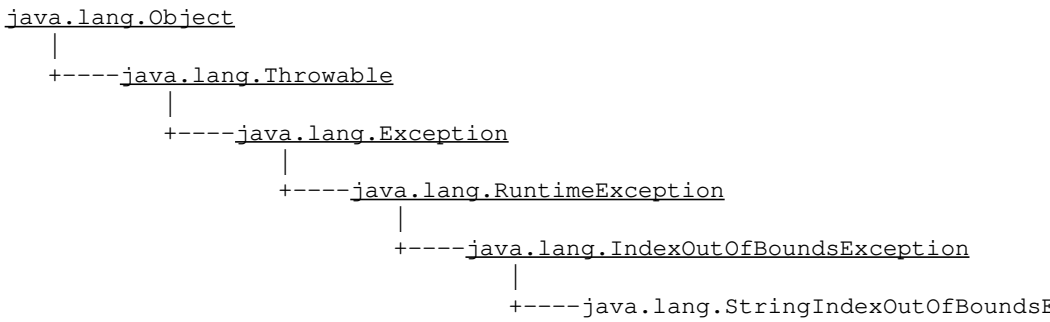
Inserts a double into the String buffer.
Parameters:
offset – the offset at which to insert
d – the double to insert
Returns:
the StringBuffer itself, NOT a new one.
Throws: StringIndexOutOfBoundsException
If the offset is invalid.

toString

```
public String toString()
```

Converts to a String representing the data in the buffer.
Overrides:
toString in class Object

Class java.lang.StringIndexOutOfBoundsException



public class **StringIndexOutOfBoundsException**
extends IndexOutOfBoundsException
Signals that a String index is out of range.
See Also:
charAt

Constructor Index

- **StringIndexOutOfBoundsException**()
  Constructs a StringIndexOutOfBoundsException with no detail message.
- **StringIndexOutOfBoundsException**(String)
  Constructs a StringIndexOutOfBoundsException with the specified detail message.
- **StringIndexOutOfBoundsException**(int)
  Constructs a StringIndexOutOfBoundsException initialized with the specified index.

Constructors

StringIndexOutOfBoundsException

```
public StringIndexOutOfBoundsException()
```

Constructs a StringIndexOutOfBoundsException with no

detail message. A detail message is a String that describes this particular exception.

StringIndexOutOfBoundsException

```
public StringIndexOutOfBoundsException(String s)
```

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

Parameters:

s – the String containing a detail message about the error

StringIndexOutOfBoundsException

```
public StringIndexOutOfBoundsException(int index)
```

Constructs a StringIndexOutOfBoundsException initialized with the specified index.

Parameters:

index – the offending index

Class java.lang.System

```
java.lang.Object
    |
    +----java.lang.System
```

---

public final class **System**
extends Object
This Class provides a system–independent interface to system functionality. One of the more useful things provided by this Class are the standard input and output streams. The standard input streams are used for reading character data. The standard output streams are used for printing. For example:

```
System.out.println("Hello World!");
```

This Class cannot be instantiated or subclassed because all of the methods and variables are static.

Variable Index

- **err**
  Standard error stream.
- **in**
  Standard input stream.
- **out**
  Standard output stream.

Method Index

- **arraycopy**(Object, int, Object, int, int)
  Copies an array from the source array, beginning at the specified position, to the specified position of the destination array.
- **currentTimeMillis**()
  Returns the current time in milliseconds GMT since the epoch (00:00:00 UTC, January 1, 1970).
- **exit**(int)
  Exits the virtual machine with an exit code.

- **gc**()
  Runs the garbage collector.
- **getProperties**()
  Gets the System properties.
- **getProperty**(String)
  Gets the System property indicated by the specified key.
- **getProperty**(String, String)
  Gets the System property indicated by the specified key and def.
- **getSecurityManager**()
  Gets the system security interface.
- **getenv**(String)
  Obsolete.
- **load**(String)
  Loads a dynamic library, given a complete path name.
- **loadLibrary**(String)
  Loads a dynamic library with the specified library name.
- **runFinalization**()
  Runs the finalization methods of any objects pending finalization.
- **setProperties**(Properties)
  Sets the System properties to the specified properties.
- **setSecurityManager**(SecurityManager)
  Sets the System security.

in

```
public static InputStream in
```

Standard input stream. This stream is used for reading in character data.

out

```
public static PrintStream out
```

Standard output stream. This stream is used for printing messages.

err

```
public static PrintStream err
```

Standard error stream. This stream can be used to print error messages. Many applications read in data from an InputStream and output messages via the PrintStream out statement. Often applications rely on command line redirection to specify source and destination files. A problem

with redirecting standard output is the incapability of writing messages to the screen if the output has been redirected to a file. This problem can be overcome by sending some output to PrintStream out and other output to PrintStream err. The difference between PrintStream err and PrintStream out is that PrintStream err is often used for displaying error messages but may be used for any purpose.

Methods

setSecurityManager

```
public static void setSecurityManager(SecurityManager s)
```

Sets the System security. This value can only be set once.

Parameters:

s – the security manager

Throws: SecurityException

If the SecurityManager has already been set.

getSecurityManager

```
public static SecurityManager getSecurityManager()
```

Gets the system security interface.

currentTimeMillis

```
public static long currentTimeMillis()
```

Returns the current time in milliseconds GMT since the epoch (00:00:00 UTC, January 1, 1970). It is a signed 64 bit integer, and so it will not overflow until the year 292280995.

See Also:

Date

arraycopy

```
public static void arraycopy(Object src,
                             int src_position,
                             Object dst,
                             int dst_position,
                             int length)
```

Copies an array from the source array, beginning at the specified position, to the specified position of the destination array. This method does not allocate memory for the destination array. The memory must already be allocated.

Parameters:

src – the source data

srcpos – start position in the source data

dest – the destination
destpos – start position in the destination data
length – the number of array elements to be copied

Throws: ArrayIndexOutOfBoundsException
If copy would cause access of data outside array bounds.

Throws: ArrayStoreException
If an element in the src array could could not be stored into the destination array due to a type mismatch

getProperties

```
public static Properties getProperties()
```

Gets the System properties.

setProperties

```
public static void setProperties(Properties props)
```

Sets the System properties to the specified properties.

Parameters:
props – the properties to be set

getProperty

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

Gets the System property indicated by the specified key.

Parameters:
key – the name of the system property

getProperty

```
public static String getProperty(String key,
                                 String def)
```

Gets the System property indicated by the specified key and def.

Parameters:
key – the name of the system property
def – the default value to use if this property is not set

getenv

```
public static String getenv(String name)
```

Obsolete. Gets an environment variable. An environment variable is a system dependent external variable that has a string value.

Parameters:
name – the name of the environment variable

Returns:
the value of the variable, or null if the variable is not

defined.

exit

```
public static void exit(int status)
```

Exits the virtual machine with an exit code. This method does not return, use with caution.

Parameters:

status – exit status, 0 if successful, other values indicate various error types.

See Also:

exit

gc

```
public static void gc()
```

Runs the garbage collector.

See Also:

gc

runFinalization

```
public static void runFinalization()
```

Runs the finalization methods of any objects pending finalization.

See Also:

gc

load

```
public static void load(String filename)
```

Loads a dynamic library, given a complete path name.

Parameters:

filename – the file to load

Throws: UnsatisfiedLinkError

If the file does not exist.

See Also:

load

loadLibrary

```
public static void loadLibrary(String libname)
```

Loads a dynamic library with the specified library name.

Parameters:

libname – the name of the library

Throws: UnsatisfiedLinkError

If the library does not exist.

See Also:

loadLibrary

Class java.lang.Thread

```
java.lang.Object
   |
   +----java.lang.Thread
```

---

public class **Thread**
extends Object
implements Runnable
A Thread is a single sequential flow of control within a process. This simply means that while executing within a program, each thread has a beginning, a sequence, a point of execution occurring at any time during runtime of the thread and of course, an ending. Thread objects are the basis for multi−threaded programming. Multi−threaded programming allows a single program to conduct concurrently running threads that perform different tasks.

To create a new thread of execution, declare a new class which is a subclass of Thread and then override the run() method with code that you want executed in this Thread. An instance of the Thread subclass should be created next with a call to the start() method following the instance. The start() method will create the thread and execute the run() method. For example:

```
class PrimeThread extends Thread {
    public void run() {
        // compute primes...
    }
}
```

To start this thread you need to do the following:

```
PrimeThread p = new PrimeThread();
p.start();
...
```

Another way to create a thread is by using the Runnable interface. This way any object that implements the Runnable interface can be run in a thread. For example:

```
class Primes implements Runnable {
    public void run() {
        // compute primes...
    }
}
```

To start this thread you need to do the following:

```
Primes p = new Primes();
new Thread(p).start();
...
```

The virtual machine runs until all Threads that are not daemon Threads have died. A Thread dies when its run() method returns, or when the stop() method is called.

When a new Thread is created, it inherits the priority and the daemon flag from its parent (i.e.: the Thread that created it).

See Also:

Runnable

Variable Index

- **MAX_PRIORITY**
  The maximum priority that a Thread can have.
- **MIN_PRIORITY**
  The minimum priority that a Thread can have.
- **NORM_PRIORITY**
  The default priority that is assigned to a Thread.

Constructor Index

- **Thread**()
  Constructs a new Thread.
- **Thread**(Runnable)
  Constructs a new Thread which applies the run() method of the specified target.
- **Thread**(ThreadGroup, Runnable)
  Constructs a new Thread in the specified Thread group that applies the run() method of the specified target.
- **Thread**(String)
  Constructs a new Thread with the specified name.
- **Thread**(ThreadGroup, String)
  Constructs a new Thread in the specified Thread group with the specified name.
- **Thread**(Runnable, String)
  Constructs a new Thread with the specified name and applies the run() method of the specified target.
- **Thread**(ThreadGroup, Runnable, String)
  Constructs a new Thread in the specified Thread group with the specified name and applies the run() method of the

specified target.

Method Index

- **activeCount**()
  Returns the current number of active Threads in this Thread group.
- **checkAccess**()
  Checks whether the current Thread is allowed to modify this Thread.
- **countStackFrames**()
  Returns the number of stack frames in this Thread.
- **currentThread**()
  Returns a reference to the currently executing Thread object.
- **destroy**()
  Destroy a thread, without any cleanup, i.e.
- **dumpStack**()
  A debugging procedure to print a stack trace for the current Thread.
- **enumerate**(Thread[])
  Copies, into the specified array, references to every active Thread in this Thread's group.
- **getName**()
  Gets and returns this Thread's name.
- **getPriority**()
  Gets and returns the Thread's priority.
- **getThreadGroup**()
  Gets and returns this Thread group.
- **interrupt**()
  Send an interrupt to a thread.
- **interrupted**()
  Ask if you have been interrupted.
- **isAlive**()
  Returns a boolean indicating if the Thread is active.
- **isDaemon**()
  Returns the daemon flag of the Thread.
- **isInterrupted**()
  Ask if another thread has been interrupted.
- **join**(long)
  Waits for this Thread to die.
- **join**(long, int)
  Waits for the Thread to die, with more precise time.
- **join**()
  Waits forever for this Thread to die.
- **resume**()
  Resumes this Thread execution.
- **run**()

The actual body of this Thread.

- **setDaemon**(boolean)
  Marks this Thread as a daemon Thread or a user Thread.
- **setName**(String)
  Sets the Thread's name.
- **setPriority**(int)
  Sets the Thread's priority.
- **sleep**(long)
  Causes the currently executing Thread to sleep for the specified number of milliseconds.
- **sleep**(long, int)
  Sleep, in milliseconds and additional nanosecond.
- **start**()
  Starts this Thread.
- **stop**()
  Stops a Thread by tossing an object.
- **stop**(Throwable)
  Stops a Thread by tossing an object.
- **suspend**()
  Suspends this Thread's execution.
- **toString**()
  Returns a String representation of the Thread, including the thread's name, priority and thread group.
- **yield**()
  Causes the currently executing Thread object to yield.

variables

- **MIN_PRIORITY**

```
public final static int MIN_PRIORITY
```

  The minimum priority that a Thread can have. The most minimal priority is equal to 1.
- **NORM_PRIORITY**

```
public final static int NORM_PRIORITY
```

  The default priority that is assigned to a Thread. The default priority is equal to 5.
- **MAX_PRIORITY**

```
public final static int MAX_PRIORITY
```

  The maximum priority that a Thread can have. The maximal priority value a Thread can have is 10.

Constructors

Thread

```
public Thread()
```

Constructs a new Thread. Threads created this way must have overridden their run() method to actually do anything. An example illustrating this method being used is shown.

```
import java.lang.*;

class plain01 implements Runnable {
  String name;
  plain01() {
    name = null;
  }
  plain01(String s) {
    name = s;
  }
  public void run() {
    if (name == null)
      System.out.println("A new thread created");
    else
      System.out.println("A new thread with name " + name + " creat
  }
}
class threadtest01 {
  public static void main(String args[] ) {
    int failed = 0 ;

   Thread t1 = new Thread();
    if(t1 != null) {
      System.out.println("new Thread() succeed");
    } else {
       System.out.println("new Thread() failed");
       failed++;
    }


}
```

Thread

```
public Thread(Runnable target)
```

Constructs a new Thread which applies the run() method of the specified target.

Parameters:

target – the object whose run() method is called

Thread

```
public Thread(ThreadGroup group,
              Runnable target)
```

Constructs a new Thread in the specified Thread group that applies the run() method of the specified target.

Parameters:

group – the Thread group

target – the object whose run() method is called

- **Thread**

```
public Thread(String name)
```

Constructs a new Thread with the specified name.

Parameters:

name – the name of the new Thread

- **Thread**

```
public Thread(ThreadGroup group,
              String name)
```

Constructs a new Thread in the specified Thread group with the specified name.

Parameters:

group – the Thread group

name – the name of the new Thread

- **Thread**

```
public Thread(Runnable target,
              String name)
```

Constructs a new Thread with the specified name and applies the run() method of the specified target.

Parameters:

target – the object whose run() method is called

name – the name of the new Thread

- **Thread**

```
public Thread(ThreadGroup group,
              Runnable target,
              String name)
```

Constructs a new Thread in the specified Thread group with the specified name and applies the run() method of the specified target.

Parameters:

group – the Thread group

target – the object whose run() method is called

name – the name of the new Thread

Methods

currentThread

```
public static Thread currentThread()
```

Returns a reference to the currently executing Thread object.

yield

```
public static void yield()
```

Causes the currently executing Thread object to yield. If there are other runnable Threads they will be scheduled next.

sleep

```
public static void sleep(long millis) throws InterruptedException
```

Causes the currently executing Thread to sleep for the specified number of milliseconds.

Parameters:

millis – the length of time to sleep in milliseconds

Throws: InterruptedException

Another thread has interrupted this thread.

sleep

```
public static void sleep(long millis,
                         int nanos) throws InterruptedException
```

Sleep, in milliseconds and additional nanosecond.

Parameters:

millis – the length of time to sleep in milliseconds

nanos – 0–999999 additional nanoseconds to sleep

Throws: InterruptedException

Another thread has interrupted this thread.

start

```
public synchronized void start()
```

Starts this Thread. This will cause the run() method to be called. This method will return immediately.

Throws: IllegalThreadStateException

If the thread was already started.

See Also:

run, stop

run

```
public void run()
```

The actual body of this Thread. This method is called after the Thread is started. You must either override this method by subclassing class Thread, or you must create the Thread with a Runnable target.

See Also:

start, stop

stop

```
public final void stop()
```

Stops a Thread by tossing an object. By default this routine tosses a new instance of ThreadDeath to the target Thread. ThreadDeath is not actually a subclass of Exception, but is a subclass of Object. Users should not normally try to catch ThreadDeath unless they must do some extraordinary cleanup operation. If ThreadDeath is caught it is important to rethrow the object so that the thread will actually die. The top−level error handler will not print out a message if ThreadDeath falls through.

See Also:

start, run

stop

```
public final synchronized void stop(Throwable o)
```

Stops a Thread by tossing an object. Normally, users should just call the stop() method without any argument. However, in some exceptional circumstances used by the stop() method to kill a Thread, another object is tossed. ThreadDeath, is not actually a subclass of Exception, but is a subclass of Throwable

Parameters:

o − the Throwable object to be thrown

See Also:

start, run

interrupt

```
public void interrupt()
```

Send an interrupt to a thread.

interrupted

```
public static boolean interrupted()
```

Ask if you have been interrupted.

isInterrupted

```
public boolean isInterrupted()
```

Ask if another thread has been interrupted.

- **destroy**

```
public void destroy()
```

Destroy a thread, without any cleanup, i.e. just toss its state; any monitors it has locked remain locked. A last resort.

- **isAlive**

```
public final boolean isAlive()
```

Returns a boolean indicating if the Thread is active. Having an active Thread means that the Thread has been started and has not been stopped.

- **suspend**

```
public final void suspend()
```

Suspends this Thread's execution.

- **resume**

```
public final void resume()
```

Resumes this Thread execution. This method is only valid after suspend() has been invoked.

- **setPriority**

```
public final void setPriority(int newPriority)
```

Sets the Thread's priority.

Throws: IllegalArgumentException

If the priority is not within the range MIN_PRIORITY, MAX_PRIORITY.

See Also:

MIN_PRIORITY, MAX_PRIORITY, getPriority

- **getPriority**

```
public final int getPriority()
```

Gets and returns the Thread's priority.

See Also:

setPriority

- **setName**

```
public final void setName(String name)
```

Sets the Thread's name.

Parameters:

name – the new name of the Thread

See Also:
getName

getName

```
public final String getName()
```

Gets and returns this Thread's name.

See Also:
setName

getThreadGroup

```
public final ThreadGroup getThreadGroup()
```

Gets and returns this Thread group.

activeCount

```
public static int activeCount()
```

Returns the current number of active Threads in this Thread group.

enumerate

```
public static int enumerate(Thread tarray[])
```

Copies, into the specified array, references to every active Thread in this Thread's group.

Returns:
the number of Threads put into the array.

countStackFrames

```
public int countStackFrames()
```

Returns the number of stack frames in this Thread. The Thread must be suspended when this method is called.

Throws: IllegalThreadStateException
If the Thread is not suspended.

join

```
public final synchronized void join(long millis) throws InterruptedExce
```

Waits for this Thread to die. A timeout in milliseconds can be specified. A timeout of 0 milliseconds means to wait forever.

Parameters:
millis – the time to wait in milliseconds

Throws: InterruptedException
Another thread has interrupted this thread.

join

```
public final synchronized void join(long millis,
                                    int nanos) throws InterruptedExcept
```

Waits for the Thread to die, with more precise time.

Throws: InterruptedException

Another thread has interrupted this thread.

join

```
public final void join() throws InterruptedException
```

Waits forever for this Thread to die.

Throws: InterruptedException

Another thread has interrupted this thread.

dumpStack

```
public static void dumpStack()
```

A debugging procedure to print a stack trace for the current Thread.

See Also:

printStackTrace

setDaemon

```
public final void setDaemon(boolean on)
```

Marks this Thread as a daemon Thread or a user Thread. When there are only daemon Threads left running in the system, Java exits.

Parameters:

on – determines whether the Thread will be a daemon Thread

Throws: IllegalThreadStateException

If the Thread is active.

See Also:

isDaemon

isDaemon

```
public final boolean isDaemon()
```

Returns the daemon flag of the Thread.

See Also:

setDaemon

checkAccess

```
public void checkAccess()
```

Checks whether the current Thread is allowed to modify this Thread.

Throws: SecurityException

If the current Thread is not allowed to access this Thread group.

toString

```
public String toString()
```

Returns a String representation of the Thread, including the thread's name, priority and thread group.

Overrides:

toString in class Object

Class java.lang.ThreadDeath

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.ThreadDeath
```

---

public class **ThreadDeath**
extends Error
An instance of ThreadDeath is thrown in the victim thread when thread.stop() is called. This is not a subclass of Exception, but rather a subclass of Error because too many people already catch Exception. Instances of this class should be caught explicitly only if you are interested in cleaning up when being asynchronously terminated. If ThreadDeath is caught, it is important to rethrow the object so that the Thread will actually die. The top−level error handler will not print out a message if ThreadDeath falls through.

Constructor Index

ThreadDeath()

Constructors

ThreadDeath

```
public ThreadDeath()
```

Class java.lang.ThreadGroup

```
java.lang.Object
    |
    +----java.lang.ThreadGroup
```

---

public class **ThreadGroup**
extends Object
A group of Threads. A Thread group can contain a set of Threads as well as a set of other Thread groups. A Thread can access its Thread group, but it can't access the parent of its Thread group. This makes it possible to encapsulate a Thread in a Thread group and stop it from manipulating Threads in the parent group.

Constructor Index

- **ThreadGroup**(String)
  Creates a new ThreadGroup.
- **ThreadGroup**(ThreadGroup, String)
  Creates a new ThreadGroup with a specified name in the specified Thread group.

Method Index

- **activeCount**()
  Returns an estimate of the number of active Threads in the Thread group.
- **activeGroupCount**()
  Returns an estimate of the number of active groups in the Thread group.
- **checkAccess**()
  Checks to see if the current Thread is allowed to modify this group.
- **destroy**()
  Destroys a Thread group.
- **enumerate**(Thread[])
  Copies, into the specified array, references to every active Thread in this Thread group.
- **enumerate**(Thread[], boolean)
  Copies, into the specified array, references to every active

Thread in this Thread group.

- **enumerate**(ThreadGroup[])
  Copies, into the specified array, references to every active Thread group in this Thread group.
- **enumerate**(ThreadGroup[], boolean)
  Copies, into the specified array, references to every active Thread group in this Thread group.
- **getMaxPriority**()
  Gets the maximum priority of the group.
- **getName**()
  Gets the name of this Thread group.
- **getParent**()
  Gets the parent of this Thread group.
- **isDaemon**()
  Returns the daemon flag of the Thread group.
- **list**()
  Lists this Thread group.
- **parentOf**(ThreadGroup)
  Checks to see if this Thread group is a parent of or is equal to another Thread group.
- **resume**()
  Resumes all the Threads in this Thread group and all of its sub groups.
- **setDaemon**(boolean)
  Changes the daemon status of this group.
- **setMaxPriority**(int)
  Sets the maximum priority of the group.
- **stop**()
  Stops all the Threads in this Thread group and all of its sub groups.
- **suspend**()
  Suspends all the Threads in this Thread group and all of its sub groups.
- **toString**()
  Returns a String representation of the Thread group.
- **uncaughtException**(Thread, Throwable)
  Called when a thread in this group exists because of an uncaught exception.

Constructors

ThreadGroup

```
public ThreadGroup(String name)
```

Creates a new ThreadGroup. Its parent will be the Thread group of the current Thread.

Parameters:

name – the name of the new Thread group created

- **ThreadGroup**

```
public ThreadGroup(ThreadGroup parent,
                   String name)
```

Creates a new ThreadGroup with a specified name in the specified Thread group.

Parameters:

parent – the specified parent Thread group

name – the name of the new Thread group being created

Throws: NullPointerException

If the given thread group is equal to null.

- **getName**

```
public final String getName()
```

Gets the name of this Thread group.

- **getParent**

```
public final ThreadGroup getParent()
```

Gets the parent of this Thread group.

- **getMaxPriority**

```
public final int getMaxPriority()
```

Gets the maximum priority of the group. Threads that are part of this group cannot have a higher priority than the maximum priority.

- **isDaemon**

```
public final boolean isDaemon()
```

Returns the daemon flag of the Thread group. A daemon Thread group is automatically destroyed when it is found empty after a Thread group or Thread is removed from it.

- **setDaemon**

```
public final void setDaemon(boolean daemon)
```

Changes the daemon status of this group.

Parameters:

daemon – the daemon boolean which is to be set.

- **setMaxPriority**

```
public final synchronized void setMaxPriority(int pri)
```

Sets the maximum priority of the group. Threads that are already in the group **can** have a higher priority than the set maximum.

Parameters:

pri – the priority of the Thread group

- **parentOf**

```
public final boolean parentOf(ThreadGroup g)
```

Checks to see if this Thread group is a parent of or is equal to another Thread group.

Parameters:

g – the Thread group to be checked

Returns:

true if this Thread group is equal to or is the parent of another Thread group; false otherwise.

- **checkAccess**

```
public final void checkAccess()
```

Checks to see if the current Thread is allowed to modify this group.

Throws: SecurityException

If the current Thread is not allowed to access this Thread group.

- **activeCount**

```
public synchronized int activeCount()
```

Returns an estimate of the number of active Threads in the Thread group.

- **enumerate**

```
public int enumerate(Thread list[])
```

Copies, into the specified array, references to every active Thread in this Thread group. You can use the activeCount() method to get an estimate of how big the array should be.

Parameters:

list – an array of Threads

Returns:

the number of Threads put into the array

- **enumerate**

```
public int enumerate(Thread list[],
                     boolean recurse)
```

Copies, into the specified array, references to every active Thread in this Thread group. You can use the activeCount() method to get an estimate of how big the array should be.

Parameters:

list – an array list of Threads
recurse – a boolean indicating whether a Thread has reapearred

Returns:

the number of Threads placed into the array.

- **activeGroupCount**

```
public synchronized int activeGroupCount()
```

Returns an estimate of the number of active groups in the Thread group.

- **enumerate**

```
public int enumerate(ThreadGroup list[])
```

Copies, into the specified array, references to every active Thread group in this Thread group. You can use the activeGroupCount() method to get an estimate of how big the array should be.

Parameters:

list – an array of Thread groups

Returns:

the number of Thread groups placed into the array.

- **enumerate**

```
public int enumerate(ThreadGroup list[],
                     boolean recurse)
```

Copies, into the specified array, references to every active Thread group in this Thread group. You can use the activeGroupCount() method to get an estimate of how big the array should be.

Parameters:

list – an array list of Thread groups
recurse – a boolean indicating if a Thread group has reappeared

Returns:

the number of Thread groups placed into the array.

- **stop**

```
public final synchronized void stop()
```

Stops all the Threads in this Thread group and all of its sub

groups.

- **suspend**

```
public final synchronized void suspend()
```

Suspends all the Threads in this Thread group and all of its sub groups.

- **resume**

```
public final synchronized void resume()
```

Resumes all the Threads in this Thread group and all of its sub groups.

- **destroy**

```
public final synchronized void destroy()
```

Destroys a Thread group. This does **NOT** stop the Threads in the Thread group.

Throws: IllegalThreadStateException

If the Thread group is not empty or if the Thread group was already destroyed.

- **list**

```
public synchronized void list()
```

Lists this Thread group. Useful for debugging only.

- **uncaughtException**

```
public void uncaughtException(Thread t,
                              Throwable e)
```

Called when a thread in this group exists because of an uncaught exception.

- **toString**

```
public String toString()
```

Returns a String representation of the Thread group.

Overrides:

toString in class Object

Class java.lang.Throwable

```
java.lang.Object
    |
    +----java.lang.Throwable
```

---

public class **Throwable**
extends Object
An object signalling that an exceptional condition has occurred. All exceptions are a subclass of Exception. An exception contains a snapshot of the execution stack, this snapshot is used to print a stack backtrace. An exception also contains a message string. Here is an example of how to catch an exception:

```
try {
    int a[] = new int[2];
    a[4];
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("an exception occurred: " + e.getMessage()
    e.printStackTrace();
}
```

Constructor Index

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

Method Index

- **fillInStackTrace**()
  Fills in the excecution stack trace.
- **getMessage**()
  Gets the detail message of the Throwable.
- **printStackTrace**()
  Prints the Throwable and the Throwable's stack trace.
- **printStackTrace**(PrintStream)

- **toString()**
  Returns a short description of the Throwable.

Constructors

- **Throwable**

```
public Throwable()
```

  Constructs a new Throwable with no detail message. The stack trace is automatically filled in.
- **Throwable**

```
public Throwable(String message)
```

  Constructs a new Throwable with the specified detail message. The stack trace is automatically filled in.
  Parameters:
    message – the detailed message

Methods

- **getMessage**

```
public String getMessage()
```

  Gets the detail message of the Throwable. A detail message is a String that describes the Throwable that has taken place.
  Returns:
    the detail message of the throwable.
- **toString**

```
public String toString()
```

  Returns a short description of the Throwable.
  Overrides:
    toString in class Object
- **printStackTrace**

```
public void printStackTrace()
```

  Prints the Throwable and the Throwable's stack trace.
- **printStackTrace**

```
public void printStackTrace(PrintStream s)
```

fillInStackTrace

```
public Throwable fillInStackTrace()
```

Fills in the excecution stack trace. This is useful only when rethrowing a Throwable. For example:

```
try {
     a = b / c;
} catch(ArithmeticThrowable e) {
     a = Number.MAX_VALUE;
     throw e.fillInStackTrace();
}
```

Returns:
the Throwable itself.
See Also:
printStackTrace

Class java.lang.UnknownError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.VirtualMachineError
                           |
                           +----java.lang.UnknownError
```

---

public class **UnknownError**
extends VirtualMachineError
Signals that an unknown but serious exception has occurred.

Constructor Index

- **UnknownError**()
  Constructs an UnknownError with no detail message.
- **UnknownError**(String)
  Constructs an UnknownError with the specified detail message.

Constructors

UnknownError

```
public UnknownError()
```

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

UnknownError

```
public UnknownError(String s)
```

Constructs an UnknownError with the specified detail message. A detail message is a String that describes this particular exception.
Parameters:

s – the detail message

Class java.lang.UnsatisfiedLinkError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
                           |
                           +----java.lang.UnsatisfiedLinkError
```

---

public class **UnsatisfiedLinkError**
extends LinkageError
Signals an unsatisfied link.
See Also:
Runtime

Constructor Index

- **UnsatisfiedLinkError**()
  Constructs an UnsatisfiedLinkError with no detail message.
- **UnsatisfiedLinkError**(String)
  Constructs an UnsatisfiedLinkError with the specified detail message.

Constructors

UnsatisfiedLinkError

```
public UnsatisfiedLinkError()
```

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

UnsatisfiedLinkError

```
public UnsatisfiedLinkError(String s)
```

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

Parameters:

s – the detail message

Class java.lang.VerifyError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.LinkageError
                           |
                           +----java.lang.VerifyError
```

---

public class **VerifyError**
extends LinkageError
Signals that a Verification Error occured.

Constructor Index

- **VerifyError**()
  Constructor.
- **VerifyError**(String)
  Constructor with a detail message.

Constructors

VerifyError

```
public VerifyError()
```

Constructor.

VerifyError

```
public VerifyError(String s)
```

Constructor with a detail message.

Class java.lang.VirtualMachineError

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Error
                   |
                   +----java.lang.VirtualMachineError
```

---

public class **VirtualMachineError**
extends Error
A VirtualMachineError indicates that the virtual machine is broken or has run out of resources.

Constructor Index

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

Constructors

VirtualMachineError

```
public VirtualMachineError()
```

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

VirtualMachineError

```
public VirtualMachineError(String s)
```

Constructs a VirtualMachineError with the specified detail message. A detail message is a String that describes this

particular exception.

Parameters:

s – the detail message