Package java.net

All Packages

Interfaces

- ContentHandlerFactory
- FileNameMap
- SocketImplFactory
- URLStreamHandlerFactory

Classes

- ContentHandler
- DatagramPacket
- DatagramSocket
- DatagramSocketImpl
- HttpURLConnection
- InetAddress
- MulticastSocket
- ServerSocket
- Socket
- SocketImpl
- URL
- URLConnection
- URLEncoder
- URLStreamHandler

Exceptions

- BindException
- ConnectException
- MalformedURLException
- NoRouteToHostException
- ProtocolException
- SocketException
- UnknownHostException
- UnknownServiceException

ServerSocket Members

Class Overview | This Package | All Packages

Constructors

Name	Description
ServerSocket(int)	Creates a server socket on a specified port.
ServerSocket(int, int)	Creates a server socket and binds it to the specified local port number.
ServerSocket(int, int, InetAddress)	Create a server with the specified port, listen backlog, and local IP address to bind to.

Methods

Name	Description
accept()	Listens for a connection to be made to this socket and accepts it.
close()	Closes this socket.
getInetAddress()	Returns the local address of this server socket.
getLocalPort()	Returns the port on which this socket is listening.
getSoTimeout()	Retrive setting for SO_TIMEOUT.
implAccept(Socket)	Subclasses of ServerSocket use this method to override accept() to return their own subclass of socket.
setSocketFactory (SocketImplFactory)	Sets the server socket implementation factory for the application.
setSoTimeout(int)	Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
toString()	Returns the implementation address and implementation port of this socket as a **String**.

Socket Members

Class Overview | This Package | All Packages

Constructors

Name	Description
Socket()	Creates an unconnected socket, with the system-default type of SocketImpl.
Socket(InetAddress, int)	Creates a stream socket and connects it to the specified port number at the specified IP address.
Socket(InetAddress, int, boolean)	Creates a socket and connects it to the specified port number at the specified IP address. **Deprecated.**
Socket(InetAddress, int, InetAddress, int)	Creates a socket and connects it to the specified remote address on the specified remote port.
Socket(SocketImpl)	Creates an unconnected Socket with a user-specified SocketImpl.
Socket(String, int)	Creates a stream socket and connects it to the specified port number on the named host.
Socket(String, int, boolean)	Creates a stream socket and connects it to the specified port number on the named host. **Deprecated.**
Socket(String, int, InetAddress, int)	Creates a socket and connects it to the specified remote host on the specified remote port.

Methods

Name	Description
close()	Closes this socket.
getInetAddress()	Returns the address to which the socket is connected.
getInputStream()	Returns an input stream for this socket.
getLocalAddress()	Gets the local address to which the socket is bound.
getLocalPort()	Returns the local port to which this socket is bound.
getOutputStream()	Returns an output stream for this socket.
getPort()	Returns the remote port to which this socket is connected.
getSoLinger()	Returns setting for SO_LINGER.
getSoTimeout()	Returns setting for SO_TIMEOUT.
getTcpNoDelay()	Tests if TCP_NODELAY is enabled.
setSocketImplFactory(SocketImplFactory)	Sets the client socket implementation factory for the application.
setSoLinger(boolean, int)	Enable/disable SO_LINGER with the specified linger time.

setSoTimeout(int)	Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
setTcpNoDelay(boolean)	Enable/disable TCP_NODELAY (disable/enable Nagle's algorithm).
toString()	Converts this socket to a **String**.

DatagramSocket Members

Class Overview | This Package | All Packages

Constructors

Name	Description
DatagramSocket()	Constructs a datagram socket and binds it to any available port on the local host machine.
DatagramSocket(int)	Constructs a datagram socket and binds it to the specified port on the local host machine.
DatagramSocket(int, InetAddress)	Creates a datagram socket, bound to the specified local address.

Methods

Name	Description
close()	Closes this datagram socket.
getLocalAddress()	Gets the local address to which the socket is bound.
getLocalPort()	Returns the port number on the local host to which this socket is bound.
getSoTimeout()	Retrive setting for SO_TIMEOUT.
receive(DatagramPacket)	Receives a datagram packet from this socket.
send(DatagramPacket)	Sends a datagram packet from this socket.
setSoTimeout(int)	Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.

DatagramPacket Members

Class Overview | This Package | All Packages

Constructors

Name	Description
DatagramPacket(byte[], int)	Constructs a **DatagramPacket** for receiving packets of length **ilength**.
DatagramPacket(byte[], int, InetAddress, int)	Constructs a datagram packet for sending packets of length **ilength** to the specified port number on the specified host.

Methods

Name	Description
getAddress()	Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
getData()	Returns the data received or the data to be sent.
getLength()	Returns the length of the data to be sent or the length of the data received.
getPort()	Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.
setAddress(InetAddress)	
setData(byte[])	
setLength(int)	
setPort(int)	

InetAddress Members

Class Overview | This Package | All Packages

Methods

Name	Description
equals(Object)	Compares this object against the specified object.
getAddress()	Returns the raw IP address of this **InetAddress** object.
getAllByName(String)	Determines all the IP addresses of a host, given the host's name.
getByName(String)	Determines the IP address of a host, given the host's name.
getHostAddress()	Returns the IP address string "%d.%d.%d.%d"
getHostName()	Returns the hostname for this address.
getLocalHost()	Returns the local host.
hashCode()	Returns a hashcode for this IP address.
isMulticastAddress()	Utility routine to check if the InetAddress is a IP multicast address.
toString()	Converts this IP address to a **String**.

BufferedReader Members

Class Overview | This Package | All Packages

Constructors

Name	Description
BufferedReader(Reader)	Create a buffering character-input stream that uses a default-sized input buffer.
BufferedReader(Reader, int)	Create a buffering character-input stream that uses an input buffer of the specified size.

Methods

Name	Description
close()	Close the stream.
mark(int)	Mark the present position in the stream.
markSupported()	Tell whether this stream supports the mark() operation, which it does.
read()	Read a single character.
read(char[], int, int)	Read characters into a portion of an array.
readLine()	Read a line of text.
ready()	Tell whether this stream is ready to be read.
reset()	Reset the stream to the most recent mark.
skip(long)	Skip characters.

DataOutputStream Members

Class Overview | This Package | All Packages

Fields

Name	Description
written	The number of bytes written to the data output stream.

Constructors

Name	Description
DataOutputStream (OutputStream)	Creates a new data output stream to write data to the specified underlying output stream.

Methods

Name	Description
flush()	Flushes this data output stream.
size()	Returns the number of bytes written to this data output stream.
write(byte[], int, int)	Writes **len** bytes from the specified byte array starting at offset **off** to the underlying output stream.
write(int)	Writes the specified byte to the underlying output stream.
writeBoolean (boolean)	Writes a **boolean** to the underlying output stream as a 1-byte value.
writeByte(int)	Writes out a **byte** to the underlying output stream as a 1-byte value.
writeBytes (String)	Writes out the string to the underlying output stream as a sequence of bytes.
writeChar(int)	Writes a **char** to the underlying output stream as a 2-byte value, high byte first.
writeChars (String)	Writes a string to the underlying output stream as a sequence of characters.
writeDouble (double)	Converts the double argument to a **long** using the **doubleToLongBits** method in class **Double**, and then writes that **long** value to the underlying output stream as an 8-byte quantity, high byte first.
writeFloat(float)	Converts the float argument to an **int** using the **floatToIntBits** method in class **Float**, and then writes that **int** value to the underlying output stream as a 4-byte quantity, high byte first.
writeInt(int)	Writes an **int** to the underlying output stream as four bytes, high byte first.
writeLong(long)	Writes a **long** to the underlying output stream as eight bytes, high byte first.
writeShort(int)	Writes a **short** to the underlying output stream as two bytes, high byte first.

writeUTF (String)	Writes a string to the underlying output stream using UTF-8 encoding in a machine-independent manner.

Class java.lang.Thread

Class Members | This Package | All Packages

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

public class Thread
extends Object
implements Runnable

A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new **Thread** object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.

When a Java Virtual Machine starts up, there is usually a single non-daemon thread (which typically calls the method named **main** of some designated class). The Java Virtual Machine continues to execute threads until either of the following occurs:

- The **exit** method of class **Runtime** has been called and the security manager has permitted the exit operation to take place.
- All threads that are not daemon threads have died, either by returning from the call to the **run** method or by performing the **stop** method.

There are two ways to create a new thread of execution. One is to declare a class to be a subclass of **Thread**. This subclass should override the **run** method of class **Thread**. An instance of the subclass can then be allocated and started. For example, a thread that computes primes larger than a stated value could be written as follows:

```
class PrimeThread extends Thread {
    long minPrime;
    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }
    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeThread p = new PrimeThread(143);
```

```
p.start();
```

The other way to create a thread is to declare a class that implements the **Runnable** interface. That class then implements the **run** method. An instance of the class can then be allocated, passed as an argument when creating **Thread**, and started. The same example in this other style looks like the following:

```
class PrimeRun implements Runnable {
    long minPrime;
    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }
    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeRun p = new PrimeRun(143);
new Thread(p).start();
```

Every thread has a name for identification purposes. More than one thread may have the same name. If a name is not specified when a thread is created, a new name is generated for it.

See Also:

Runnable, exit, run, stop

Thread Members

Class Overview | This Package | All Packages

Fields

Name	Description
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.

Constructors

Name	Description
Thread()	Allocates a new **Thread** object.
Thread(Runnable)	Allocates a new **Thread** object.
Thread(Runnable, String)	Allocates a new **Thread** object.
Thread(String)	Allocates a new **Thread** object.
Thread(ThreadGroup, Runnable)	Allocates a new **Thread** object.
Thread(ThreadGroup, Runnable, String)	Allocates a new **Thread** object so that it has **target** as its run object, has the specified **name** as its name, and belongs to the thread group referred to by **group**.
Thread(ThreadGroup, String)	Allocates a new **Thread** object.

Methods

Name	Description
activeCount()	Returns the current number of active threads in this thread group.
checkAccess()	Determines if the currently running thread has permission to modify this thread.
countStackFrames()	Counts the number of stack frames in this thread.
currentThread()	Returns a reference to the currently executing thread object.
destroy()	Destroys this thread, without any cleanup.
dumpStack()	Prints a stack trace of the current thread.
enumerate(Thread[])	Copies into the specified array every active thread in this thread group and its subgroups.
getName()	Returns this thread's name.
getPriority()	Returns this thread's priority.

getThreadGroup()	Returns this thread's thread group.
interrupt()	Interrupts this thread.
interrupted()	Tests if the current thread has been interrupted.
isAlive()	Tests if this thread is alive.
isDaemon()	Tests if this thread is a daemon thread.
isInterrupted()	Tests if the current thread has been interrupted.
join()	Waits for this thread to die.
join(long)	Waits at most **millis** milliseconds for this thread to die.
join(long, int)	Waits at most **millis** milliseconds plus **nanos** nanoseconds for this thread to die.
resume()	Resumes a suspended thread.
run()	If this thread was constructed using a separate **Runnable** run object, then that **Runnable** object's **run** method is called; otherwise, this method does nothing and returns.
setDaemon (boolean)	Marks this thread as either a daemon thread or a user thread.
setName(String)	Changes the name of this thread to be equal to the argument **name**.
setPriority(int)	Changes the priority of this thread.
sleep(long)	Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
sleep(long, int)	Causes the currently executing thread to sleep (cease execution) for the specified number of milliseconds plus the specified number of nanoseconds.
start()	Causes this thread to begin execution; the Java Virtual Machine calls the **run** method of this thread.
stop()	Forces the thread to stop executing.
stop(Throwable)	Forces the thread to stop executing.
suspend()	Suspends this thread.
toString()	Returns a string representation of this thread, including the thread's name, priority, and thread group.
yield()	Causes the currently executing thread object to temporarily pause and allow other threads to execute.

Class java.lang.String

Class Members | This Package | All Packages

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

public final class String
extends Object
implements Serializable

The **String** class represents character strings. All string literals in Java programs, such as **"abc"**, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. 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);
```

The class **String** includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

The Java language provides special support for the string concatentation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the **StringBuffer** class and its **append** method. String conversions are implemented through the method **toString**, defined by **Object** and inherited by all classes in Java. For additional information on string concatenation and conversion, see Gosling, Joy, and Steele, *The Java Language Specification*.

See Also:
toString, StringBuffer, append, append, append, append, append, append, append, append, append, append

---

String Members

Class Overview | This Package | All Packages

Constructors

Name	Description
String()	Allocates a new **String** containing no characters.
String(byte[])	Construct a new **String** by converting the specified array of bytes using the platform's default character encoding.
String(byte[], int)	Allocates a new **String** containing characters constructed from an array of 8-bit integer values. **Deprecated.**
String(byte[], int, int)	Construct a new **String** by converting the specified subarray of bytes using the platform's default character encoding.
String(byte[], int, int, int)	Allocates a new **String** constructed from a subarray of an array of 8-bit integer values. **Deprecated.**
String(byte[], int, int, String)	Construct a new **String** by converting the specified subarray of bytes using the specified character encoding.
String(byte[], String)	Construct a new **String** by converting the specified array of bytes using the specified character encoding.
String(char[])	Allocates a new **String** so that it represents the sequence of characters currently contained in the character array argument.
String(char[], int, int)	Allocates a new **String** that contains characters from a subarray of the character array argument.
String(String)	Allocates a new string that contains the same sequence of characters as the string argument.
String(StringBuffer)	Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

Methods

Name	Description
charAt(int)	Returns the character at the specified index.
compareTo(String)	Compares two strings lexicographically.
concat(String)	Concatenates the specified string to the end of this string.
copyValueOf(char[])	Returns a String that is equivalent to the specified character array.
copyValueOf(char[], int, int)	Returns a String that is equivalent to the specified character array.
endsWith(String)	Tests if this string ends with the specified suffix.
equals(Object)	Compares this string to the specified object.
equalsIgnoreCase(String)	Compares this String to another object.

getBytes()	Convert this **String** into bytes according to the platform's default character encoding, storing the result into a new byte array.
getBytes(int, int, byte[], int)	Copies characters from this string into the destination byte array. **Deprecated.**
getBytes(String)	Convert this **String** into bytes according to the specified character encoding, storing the result into a new byte array.
getChars(int, int, char[], int)	Copies characters from this string into the destination 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 the specified index.
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, starting at the specified index.
intern()	Returns a canonical representation for the string object.
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, searching backward starting at the specified index.
lastIndexOf(String)	Returns the index within this string of the rightmost 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 this string.
regionMatches(boolean, int, String, int, int)	Tests if two string regions are equal.
regionMatches(int, String, int, int)	Tests if two string regions are equal.
replace(char, char)	Returns a new string resulting from replacing all occurrences of **oldChar** in this string with **newChar**.
startsWith(String)	Tests if this string starts with the specified prefix.
startsWith(String, int)	Tests if this string starts with the specified prefix.
substring(int)	Returns a new string that is a substring of this string.
substring(int, int)	Returns a new string that is a substring of this string.
toCharArray()	Converts this string to a new character array.

toLowerCase()	Converts this **String** to lowercase.
toLowerCase(Locale)	Converts all of the characters in this **String** to lower case using the rules of the given locale.
toString()	This object (which is already a string!) is itself returned.
toUpperCase()	Converts this string to uppercase.
toUpperCase(Locale)	Converts all of the characters in this **String** to upper case using the rules of the given locale.
trim()	Removes white space from both ends of this string.
valueOf(boolean)	Returns the string representation of the **boolean** argument.
valueOf(char)	Returns the string representation of the **char** * argument.
valueOf(char[])	Returns the string representation of the **char** array argument.
valueOf(char[], int, int)	Returns the string representation of a specific subarray of the **char** array argument.
valueOf(double)	Returns the string representation of the **double** argument.
valueOf(float)	Returns the string representation of the **float** argument.
valueOf(int)	Returns the string representation of the **int** argument.
valueOf(long)	Returns the string representation of the **long** argument.
valueOf(Object)	Returns the string representation of the **Object** argument.

Class java.util.StringTokenizer

Class Members | This Package | All Packages

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

public class StringTokenizer
extends Object
implements Enumeration

The string tokenizer class allows an application to break a string into tokens. The tokenization method is much simpler than the one used by the **StreamTokenizer** class. The **StringTokenizer** methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments.

The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.

An instance of **StringTokenizer** behaves in one of two ways, depending on whether it was created with the **returnTokens** flag having the value **true** or **false**:

- If the flag is **false**, delimiter characters serve to separate tokens. A token is a maximal sequence of consecutive characters that are not delimiters.
- If the flag is **true**, delimiter characters are considered to be tokens. A token is either one delimiter character, or a maximal sequence of consecutive characters that are not delimiters.

The following is one example of the use of the tokenizer. The code:

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

prints the following output:

```
this
is
a
test
```

See Also:
StreamTokenizer

StringTokenizer Members

Class Overview | This Package | All Packages

Constructors

Name	Description
StringTokenizer(String)	Constructs a string tokenizer for the specified string.
StringTokenizer(String, String)	Constructs a string tokenizer for the specified string.
StringTokenizer(String, String, boolean)	Constructs a string tokenizer for the specified string.

Methods

Name	Description
countTokens()	Calculates the number of times that this tokenizer's **nextToken** method can be called before it generates an exception.
hasMoreElements()	Returns the same value as the **hasMoreTokens** method.
hasMoreTokens()	Tests if there are more tokens available from this tokenizer's string.
nextElement()	Returns the same value as the **nextToken** method, except that its declared return value is **Object** rather than **String**.
nextToken()	Returns the next token from this string tokenizer.
nextToken(String)	Returns the next token in this string tokenizer's string.