| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/SecurityManager.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/lang/SecurityException.html)   [**NEXT CLASS**](http://docs.google.com/java/lang/Short.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/SecurityManager.html)    [**NO FRAMES**](http://docs.google.com/SecurityManager.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#2s8eyo1) | [METHOD](#3rdcrjn) |

## **java.lang**

Class SecurityManager

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **java.lang.SecurityManager**

**Direct Known Subclasses:** [RMISecurityManager](http://docs.google.com/java/rmi/RMISecurityManager.html)

public class **SecurityManager**extends [Object](http://docs.google.com/java/lang/Object.html)

The security manager is a class that allows applications to implement a security policy. It allows an application to determine, before performing a possibly unsafe or sensitive operation, what the operation is and whether it is being attempted in a security context that allows the operation to be performed. The application can allow or disallow the operation.

The SecurityManager class contains many methods with names that begin with the word check. These methods are called by various methods in the Java libraries before those methods perform certain potentially sensitive operations. The invocation of such a check method typically looks like this:

SecurityManager security = System.getSecurityManager();  
 if (security != null) {  
 security.check*XXX*(argument,  . . . );  
 }

The security manager is thereby given an opportunity to prevent completion of the operation by throwing an exception. A security manager routine simply returns if the operation is permitted, but throws a SecurityException if the operation is not permitted. The only exception to this convention is checkTopLevelWindow, which returns a boolean value.

The current security manager is set by the setSecurityManager method in class System. The current security manager is obtained by the getSecurityManager method.

The special method [checkPermission(java.security.Permission)](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission)) determines whether an access request indicated by a specified permission should be granted or denied. The default implementation calls

AccessController.checkPermission(perm);

If a requested access is allowed, checkPermission returns quietly. If denied, a SecurityException is thrown.

As of Java 2 SDK v1.2, the default implementation of each of the other check methods in SecurityManager is to call the SecurityManager checkPermission method to determine if the calling thread has permission to perform the requested operation.

Note that the checkPermission method with just a single permission argument always performs security checks within the context of the currently executing thread. Sometimes a security check that should be made within a given context will actually need to be done from within a *different* context (for example, from within a worker thread). The [getSecurityContext](http://docs.google.com/java/lang/SecurityManager.html#getSecurityContext()) method and the [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission,%20java.lang.Object)) method that includes a context argument are provided for this situation. The getSecurityContext method returns a "snapshot" of the current calling context. (The default implementation returns an AccessControlContext object.) A sample call is the following:

Object context = null;  
 SecurityManager sm = System.getSecurityManager();  
 if (sm != null) context = sm.getSecurityContext();

The checkPermission method that takes a context object in addition to a permission makes access decisions based on that context, rather than on that of the current execution thread. Code within a different context can thus call that method, passing the permission and the previously-saved context object. A sample call, using the SecurityManager sm obtained as in the previous example, is the following:

if (sm != null) sm.checkPermission(permission, context);

Permissions fall into these categories: File, Socket, Net, Security, Runtime, Property, AWT, Reflect, and Serializable. The classes managing these various permission categories are java.io.FilePermission, java.net.SocketPermission, java.net.NetPermission, java.security.SecurityPermission, java.lang.RuntimePermission, java.util.PropertyPermission, java.awt.AWTPermission, java.lang.reflect.ReflectPermission, and java.io.SerializablePermission.

All but the first two (FilePermission and SocketPermission) are subclasses of java.security.BasicPermission, which itself is an abstract subclass of the top-level class for permissions, which is java.security.Permission. BasicPermission defines the functionality needed for all permissions that contain a name that follows the hierarchical property naming convention (for example, "exitVM", "setFactory", "queuePrintJob", etc). An asterisk may appear at the end of the name, following a ".", or by itself, to signify a wildcard match. For example: "a.\*" or "\*" is valid, "\*a" or "a\*b" is not valid.

FilePermission and SocketPermission are subclasses of the top-level class for permissions (java.security.Permission). Classes like these that have a more complicated name syntax than that used by BasicPermission subclass directly from Permission rather than from BasicPermission. For example, for a java.io.FilePermission object, the permission name is the path name of a file (or directory).

Some of the permission classes have an "actions" list that tells the actions that are permitted for the object. For example, for a java.io.FilePermission object, the actions list (such as "read, write") specifies which actions are granted for the specified file (or for files in the specified directory).

Other permission classes are for "named" permissions - ones that contain a name but no actions list; you either have the named permission or you don't.

Note: There is also a java.security.AllPermission permission that implies all permissions. It exists to simplify the work of system administrators who might need to perform multiple tasks that require all (or numerous) permissions.

See  [Permissions in the JDK](http://docs.google.com/technotes/guides/security/permissions.html) for permission-related information. This document includes, for example, a table listing the various SecurityManager check methods and the permission(s) the default implementation of each such method requires. It also contains a table of all the version 1.2 methods that require permissions, and for each such method tells which permission it requires.

For more information about SecurityManager changes made in the JDK and advice regarding porting of 1.1-style security managers, see the [security documentation](http://docs.google.com/technotes/guides/security/index.html).

**Since:** JDK1.0 **See Also:**[ClassLoader](http://docs.google.com/java/lang/ClassLoader.html), [SecurityException](http://docs.google.com/java/lang/SecurityException.html), [checkTopLevelWindow](http://docs.google.com/java/lang/SecurityManager.html#checkTopLevelWindow(java.lang.Object)), [getSecurityManager](http://docs.google.com/java/lang/System.html#getSecurityManager()), [setSecurityManager](http://docs.google.com/java/lang/System.html#setSecurityManager(java.lang.SecurityManager)), [AccessController](http://docs.google.com/java/security/AccessController.html), [AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html), [AccessControlException](http://docs.google.com/java/security/AccessControlException.html), [Permission](http://docs.google.com/java/security/Permission.html), [BasicPermission](http://docs.google.com/java/security/BasicPermission.html), [FilePermission](http://docs.google.com/java/io/FilePermission.html), [SocketPermission](http://docs.google.com/java/net/SocketPermission.html), [PropertyPermission](http://docs.google.com/java/util/PropertyPermission.html), [RuntimePermission](http://docs.google.com/java/lang/RuntimePermission.html), [AWTPermission](http://docs.google.com/java/awt/AWTPermission.html), [Policy](http://docs.google.com/java/security/Policy.html), [SecurityPermission](http://docs.google.com/java/security/SecurityPermission.html), [ProtectionDomain](http://docs.google.com/java/security/ProtectionDomain.html)

| **Field Summary** | |
| --- | --- |
| protected  boolean | [**inCheck**](http://docs.google.com/java/lang/SecurityManager.html#inCheck)  **Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.* |

| **Constructor Summary** | |
| --- | --- |
| [**SecurityManager**](http://docs.google.com/java/lang/SecurityManager.html#SecurityManager())()            Constructs a new SecurityManager. |

| **Method Summary** | |
| --- | --- |
| void | [**checkAccept**](http://docs.google.com/java/lang/SecurityManager.html#checkAccept(java.lang.String,%20int))([String](http://docs.google.com/java/lang/String.html) host, int port)            Throws a SecurityException if the calling thread is not permitted to accept a socket connection from the specified host and port number. |
| void | [**checkAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkAccess(java.lang.Thread))([Thread](http://docs.google.com/java/lang/Thread.html) t)            Throws a SecurityException if the calling thread is not allowed to modify the thread argument. |
| void | [**checkAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkAccess(java.lang.ThreadGroup))([ThreadGroup](http://docs.google.com/java/lang/ThreadGroup.html) g)            Throws a SecurityException if the calling thread is not allowed to modify the thread group argument. |
| void | [**checkAwtEventQueueAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkAwtEventQueueAccess())()            Throws a SecurityException if the calling thread is not allowed to access the AWT event queue. |
| void | [**checkConnect**](http://docs.google.com/java/lang/SecurityManager.html#checkConnect(java.lang.String,%20int))([String](http://docs.google.com/java/lang/String.html) host, int port)            Throws a SecurityException if the calling thread is not allowed to open a socket connection to the specified host and port number. |
| void | [**checkConnect**](http://docs.google.com/java/lang/SecurityManager.html#checkConnect(java.lang.String,%20int,%20java.lang.Object))([String](http://docs.google.com/java/lang/String.html) host, int port, [Object](http://docs.google.com/java/lang/Object.html) context)            Throws a SecurityException if the specified security context is not allowed to open a socket connection to the specified host and port number. |
| void | [**checkCreateClassLoader**](http://docs.google.com/java/lang/SecurityManager.html#checkCreateClassLoader())()            Throws a SecurityException if the calling thread is not allowed to create a new class loader. |
| void | [**checkDelete**](http://docs.google.com/java/lang/SecurityManager.html#checkDelete(java.lang.String))([String](http://docs.google.com/java/lang/String.html) file)            Throws a SecurityException if the calling thread is not allowed to delete the specified file. |
| void | [**checkExec**](http://docs.google.com/java/lang/SecurityManager.html#checkExec(java.lang.String))([String](http://docs.google.com/java/lang/String.html) cmd)            Throws a SecurityException if the calling thread is not allowed to create a subprocess. |
| void | [**checkExit**](http://docs.google.com/java/lang/SecurityManager.html#checkExit(int))(int status)            Throws a SecurityException if the calling thread is not allowed to cause the Java Virtual Machine to halt with the specified status code. |
| void | [**checkLink**](http://docs.google.com/java/lang/SecurityManager.html#checkLink(java.lang.String))([String](http://docs.google.com/java/lang/String.html) lib)            Throws a SecurityException if the calling thread is not allowed to dynamic link the library code specified by the string argument file. |
| void | [**checkListen**](http://docs.google.com/java/lang/SecurityManager.html#checkListen(int))(int port)            Throws a SecurityException if the calling thread is not allowed to wait for a connection request on the specified local port number. |
| void | [**checkMemberAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkMemberAccess(java.lang.Class,%20int))([Class](http://docs.google.com/java/lang/Class.html)<?> clazz, int which)            Throws a SecurityException if the calling thread is not allowed to access members. |
| void | [**checkMulticast**](http://docs.google.com/java/lang/SecurityManager.html#checkMulticast(java.net.InetAddress))([InetAddress](http://docs.google.com/java/net/InetAddress.html) maddr)            Throws a SecurityException if the calling thread is not allowed to use (join/leave/send/receive) IP multicast. |
| void | [**checkMulticast**](http://docs.google.com/java/lang/SecurityManager.html#checkMulticast(java.net.InetAddress,%20byte))([InetAddress](http://docs.google.com/java/net/InetAddress.html) maddr, byte ttl)  **Deprecated.** *Use #checkPermission(java.security.Permission) instead* |
| void | [**checkPackageAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkPackageAccess(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pkg)            Throws a SecurityException if the calling thread is not allowed to access the package specified by the argument. |
| void | [**checkPackageDefinition**](http://docs.google.com/java/lang/SecurityManager.html#checkPackageDefinition(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pkg)            Throws a SecurityException if the calling thread is not allowed to define classes in the package specified by the argument. |
| void | [**checkPermission**](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))([Permission](http://docs.google.com/java/security/Permission.html) perm)            Throws a SecurityException if the requested access, specified by the given permission, is not permitted based on the security policy currently in effect. |
| void | [**checkPermission**](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission,%20java.lang.Object))([Permission](http://docs.google.com/java/security/Permission.html) perm, [Object](http://docs.google.com/java/lang/Object.html) context)            Throws a SecurityException if the specified security context is denied access to the resource specified by the given permission. |
| void | [**checkPrintJobAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkPrintJobAccess())()            Throws a SecurityException if the calling thread is not allowed to initiate a print job request. |
| void | [**checkPropertiesAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkPropertiesAccess())()            Throws a SecurityException if the calling thread is not allowed to access or modify the system properties. |
| void | [**checkPropertyAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkPropertyAccess(java.lang.String))([String](http://docs.google.com/java/lang/String.html) key)            Throws a SecurityException if the calling thread is not allowed to access the system property with the specified key name. |
| void | [**checkRead**](http://docs.google.com/java/lang/SecurityManager.html#checkRead(java.io.FileDescriptor))([FileDescriptor](http://docs.google.com/java/io/FileDescriptor.html) fd)            Throws a SecurityException if the calling thread is not allowed to read from the specified file descriptor. |
| void | [**checkRead**](http://docs.google.com/java/lang/SecurityManager.html#checkRead(java.lang.String))([String](http://docs.google.com/java/lang/String.html) file)            Throws a SecurityException if the calling thread is not allowed to read the file specified by the string argument. |
| void | [**checkRead**](http://docs.google.com/java/lang/SecurityManager.html#checkRead(java.lang.String,%20java.lang.Object))([String](http://docs.google.com/java/lang/String.html) file, [Object](http://docs.google.com/java/lang/Object.html) context)            Throws a SecurityException if the specified security context is not allowed to read the file specified by the string argument. |
| void | [**checkSecurityAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkSecurityAccess(java.lang.String))([String](http://docs.google.com/java/lang/String.html) target)            Determines whether the permission with the specified permission target name should be granted or denied. |
| void | [**checkSetFactory**](http://docs.google.com/java/lang/SecurityManager.html#checkSetFactory())()            Throws a SecurityException if the calling thread is not allowed to set the socket factory used by ServerSocket or Socket, or the stream handler factory used by URL. |
| void | [**checkSystemClipboardAccess**](http://docs.google.com/java/lang/SecurityManager.html#checkSystemClipboardAccess())()            Throws a SecurityException if the calling thread is not allowed to access the system clipboard. |
| boolean | [**checkTopLevelWindow**](http://docs.google.com/java/lang/SecurityManager.html#checkTopLevelWindow(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) window)            Returns false if the calling thread is not trusted to bring up the top-level window indicated by the window argument. |
| void | [**checkWrite**](http://docs.google.com/java/lang/SecurityManager.html#checkWrite(java.io.FileDescriptor))([FileDescriptor](http://docs.google.com/java/io/FileDescriptor.html) fd)            Throws a SecurityException if the calling thread is not allowed to write to the specified file descriptor. |
| void | [**checkWrite**](http://docs.google.com/java/lang/SecurityManager.html#checkWrite(java.lang.String))([String](http://docs.google.com/java/lang/String.html) file)            Throws a SecurityException if the calling thread is not allowed to write to the file specified by the string argument. |
| protected  int | [**classDepth**](http://docs.google.com/java/lang/SecurityManager.html#classDepth(java.lang.String))([String](http://docs.google.com/java/lang/String.html) name)  **Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.* |
| protected  int | [**classLoaderDepth**](http://docs.google.com/java/lang/SecurityManager.html#classLoaderDepth())()  **Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.* |
| protected  [ClassLoader](http://docs.google.com/java/lang/ClassLoader.html) | [**currentClassLoader**](http://docs.google.com/java/lang/SecurityManager.html#currentClassLoader())()  **Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.* |
| protected  [Class](http://docs.google.com/java/lang/Class.html)<?> | [**currentLoadedClass**](http://docs.google.com/java/lang/SecurityManager.html#currentLoadedClass())()  **Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.* |
| protected  [Class](http://docs.google.com/java/lang/Class.html)[] | [**getClassContext**](http://docs.google.com/java/lang/SecurityManager.html#getClassContext())()            Returns the current execution stack as an array of classes. |
| boolean | [**getInCheck**](http://docs.google.com/java/lang/SecurityManager.html#getInCheck())()  **Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.* |
| [Object](http://docs.google.com/java/lang/Object.html) | [**getSecurityContext**](http://docs.google.com/java/lang/SecurityManager.html#getSecurityContext())()            Creates an object that encapsulates the current execution environment. |
| [ThreadGroup](http://docs.google.com/java/lang/ThreadGroup.html) | [**getThreadGroup**](http://docs.google.com/java/lang/SecurityManager.html#getThreadGroup())()            Returns the thread group into which to instantiate any new thread being created at the time this is being called. |
| protected  boolean | [**inClass**](http://docs.google.com/java/lang/SecurityManager.html#inClass(java.lang.String))([String](http://docs.google.com/java/lang/String.html) name)  **Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.* |
| protected  boolean | [**inClassLoader**](http://docs.google.com/java/lang/SecurityManager.html#inClassLoader())()  **Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.* |

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

| **Field Detail** |
| --- |

### inCheck

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
protected boolean **inCheck**

**Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.*This field is true if there is a security check in progress; false otherwise.

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

### SecurityManager

public **SecurityManager**()

Constructs a new SecurityManager.

If there is a security manager already installed, this method first calls the security manager's checkPermission method with the RuntimePermission("createSecurityManager") permission to ensure the calling thread has permission to create a new security manager. This may result in throwing a SecurityException.

**Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if a security manager already exists and its checkPermission method doesn't allow creation of a new security manager.**See Also:**[System.getSecurityManager()](http://docs.google.com/java/lang/System.html#getSecurityManager()), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission)), [RuntimePermission](http://docs.google.com/java/lang/RuntimePermission.html)

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

### getInCheck

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public boolean **getInCheck**()

**Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.*

Tests if there is a security check in progress.

**Returns:**the value of the inCheck field. This field should contain true if a security check is in progress, false otherwise.**See Also:**[inCheck](http://docs.google.com/java/lang/SecurityManager.html#inCheck)

### getClassContext

protected [Class](http://docs.google.com/java/lang/Class.html)[] **getClassContext**()

Returns the current execution stack as an array of classes.

The length of the array is the number of methods on the execution stack. The element at index 0 is the class of the currently executing method, the element at index 1 is the class of that method's caller, and so on.

**Returns:**the execution stack.

### currentClassLoader

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
protected [ClassLoader](http://docs.google.com/java/lang/ClassLoader.html) **currentClassLoader**()

**Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.*

Returns the class loader of the most recently executing method from a class defined using a non-system class loader. A non-system class loader is defined as being a class loader that is not equal to the system class loader (as returned by [ClassLoader.getSystemClassLoader()](http://docs.google.com/java/lang/ClassLoader.html#getSystemClassLoader())) or one of its ancestors.

This method will return null in the following three cases:

1. All methods on the execution stack are from classes defined using the system class loader or one of its ancestors.
2. All methods on the execution stack up to the first "privileged" caller (see [AccessController.doPrivileged(java.security.PrivilegedAction)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction))) are from classes defined using the system class loader or one of its ancestors.
3. A call to checkPermission with java.security.AllPermission does not result in a SecurityException.

**Returns:**the class loader of the most recent occurrence on the stack of a method from a class defined using a non-system class loader.**See Also:**[getSystemClassLoader](http://docs.google.com/java/lang/ClassLoader.html#getSystemClassLoader()), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### currentLoadedClass

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
protected [Class](http://docs.google.com/java/lang/Class.html)<?> **currentLoadedClass**()

**Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.*

Returns the class of the most recently executing method from a class defined using a non-system class loader. A non-system class loader is defined as being a class loader that is not equal to the system class loader (as returned by [ClassLoader.getSystemClassLoader()](http://docs.google.com/java/lang/ClassLoader.html#getSystemClassLoader())) or one of its ancestors.

This method will return null in the following three cases:

1. All methods on the execution stack are from classes defined using the system class loader or one of its ancestors.
2. All methods on the execution stack up to the first "privileged" caller (see [AccessController.doPrivileged(java.security.PrivilegedAction)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction))) are from classes defined using the system class loader or one of its ancestors.
3. A call to checkPermission with java.security.AllPermission does not result in a SecurityException.

**Returns:**the class of the most recent occurrence on the stack of a method from a class defined using a non-system class loader.**See Also:**[getSystemClassLoader](http://docs.google.com/java/lang/ClassLoader.html#getSystemClassLoader()), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### classDepth

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
protected int **classDepth**([String](http://docs.google.com/java/lang/String.html) name)

**Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.*

Returns the stack depth of the specified class.

**Parameters:**name - the fully qualified name of the class to search for. **Returns:**the depth on the stack frame of the first occurrence of a method from a class with the specified name; -1 if such a frame cannot be found.

### classLoaderDepth

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
protected int **classLoaderDepth**()

**Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.*

Returns the stack depth of the most recently executing method from a class defined using a non-system class loader. A non-system class loader is defined as being a class loader that is not equal to the system class loader (as returned by [ClassLoader.getSystemClassLoader()](http://docs.google.com/java/lang/ClassLoader.html#getSystemClassLoader())) or one of its ancestors.

This method will return -1 in the following three cases:

1. All methods on the execution stack are from classes defined using the system class loader or one of its ancestors.
2. All methods on the execution stack up to the first "privileged" caller (see [AccessController.doPrivileged(java.security.PrivilegedAction)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction))) are from classes defined using the system class loader or one of its ancestors.
3. A call to checkPermission with java.security.AllPermission does not result in a SecurityException.

**Returns:**the depth on the stack frame of the most recent occurrence of a method from a class defined using a non-system class loader.**See Also:**[getSystemClassLoader](http://docs.google.com/java/lang/ClassLoader.html#getSystemClassLoader()), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### inClass

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
protected boolean **inClass**([String](http://docs.google.com/java/lang/String.html) name)

**Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.*

Tests if a method from a class with the specified name is on the execution stack.

**Parameters:**name - the fully qualified name of the class. **Returns:**true if a method from a class with the specified name is on the execution stack; false otherwise.

### inClassLoader

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
protected boolean **inClassLoader**()

**Deprecated.** *This type of security checking is not recommended. It is recommended that the checkPermission call be used instead.*

Basically, tests if a method from a class defined using a class loader is on the execution stack.

**Returns:**true if a call to currentClassLoader has a non-null return value.**See Also:**[currentClassLoader](http://docs.google.com/java/lang/SecurityManager.html#currentClassLoader())

### getSecurityContext

public [Object](http://docs.google.com/java/lang/Object.html) **getSecurityContext**()

Creates an object that encapsulates the current execution environment. The result of this method is used, for example, by the three-argument checkConnect method and by the two-argument checkRead method. These methods are needed because a trusted method may be called on to read a file or open a socket on behalf of another method. The trusted method needs to determine if the other (possibly untrusted) method would be allowed to perform the operation on its own.

The default implementation of this method is to return an AccessControlContext object.

**Returns:**an implementation-dependent object that encapsulates sufficient information about the current execution environment to perform some security checks later.**See Also:**[checkConnect](http://docs.google.com/java/lang/SecurityManager.html#checkConnect(java.lang.String,%20int,%20java.lang.Object)), [checkRead](http://docs.google.com/java/lang/SecurityManager.html#checkRead(java.lang.String,%20java.lang.Object)), [AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html)

### checkPermission

public void **checkPermission**([Permission](http://docs.google.com/java/security/Permission.html) perm)

Throws a SecurityException if the requested access, specified by the given permission, is not permitted based on the security policy currently in effect.

This method calls AccessController.checkPermission with the given permission.

**Parameters:**perm - the requested permission. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if access is not permitted based on the current security policy. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the permission argument is null.**Since:** 1.2

### checkPermission

public void **checkPermission**([Permission](http://docs.google.com/java/security/Permission.html) perm,  
 [Object](http://docs.google.com/java/lang/Object.html) context)

Throws a SecurityException if the specified security context is denied access to the resource specified by the given permission. The context must be a security context returned by a previous call to getSecurityContext and the access control decision is based upon the configured security policy for that security context.

If context is an instance of AccessControlContext then the AccessControlContext.checkPermission method is invoked with the specified permission.

If context is not an instance of AccessControlContext then a SecurityException is thrown.

**Parameters:**perm - the specified permissioncontext - a system-dependent security context. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the specified security context is not an instance of AccessControlContext (e.g., is null), or is denied access to the resource specified by the given permission. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the permission argument is null.**Since:** 1.2 **See Also:**[getSecurityContext()](http://docs.google.com/java/lang/SecurityManager.html#getSecurityContext()), [AccessControlContext.checkPermission(java.security.Permission)](http://docs.google.com/java/security/AccessControlContext.html#checkPermission(java.security.Permission))

### checkCreateClassLoader

public void **checkCreateClassLoader**()

Throws a SecurityException if the calling thread is not allowed to create a new class loader.

This method calls checkPermission with the RuntimePermission("createClassLoader") permission.

If you override this method, then you should make a call to super.checkCreateClassLoader at the point the overridden method would normally throw an exception.

**Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to create a new class loader.**See Also:**[ClassLoader.ClassLoader()](http://docs.google.com/java/lang/ClassLoader.html#ClassLoader()), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkAccess

public void **checkAccess**([Thread](http://docs.google.com/java/lang/Thread.html) t)

Throws a SecurityException if the calling thread is not allowed to modify the thread argument.

This method is invoked for the current security manager by the stop, suspend, resume, setPriority, setName, and setDaemon methods of class Thread.

If the thread argument is a system thread (belongs to the thread group with a null parent) then this method calls checkPermission with the RuntimePermission("modifyThread") permission. If the thread argument is *not* a system thread, this method just returns silently.

Applications that want a stricter policy should override this method. If this method is overridden, the method that overrides it should additionally check to see if the calling thread has the RuntimePermission("modifyThread") permission, and if so, return silently. This is to ensure that code granted that permission (such as the JDK itself) is allowed to manipulate any thread.

If this method is overridden, then super.checkAccess should be called by the first statement in the overridden method, or the equivalent security check should be placed in the overridden method.

**Parameters:**t - the thread to be checked. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to modify the thread. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the thread argument is null.**See Also:**[resume](http://docs.google.com/java/lang/Thread.html#resume()), [setDaemon](http://docs.google.com/java/lang/Thread.html#setDaemon(boolean)), [setName](http://docs.google.com/java/lang/Thread.html#setName(java.lang.String)), [setPriority](http://docs.google.com/java/lang/Thread.html#setPriority(int)), [stop](http://docs.google.com/java/lang/Thread.html#stop()), [suspend](http://docs.google.com/java/lang/Thread.html#suspend()), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkAccess

public void **checkAccess**([ThreadGroup](http://docs.google.com/java/lang/ThreadGroup.html) g)

Throws a SecurityException if the calling thread is not allowed to modify the thread group argument.

This method is invoked for the current security manager when a new child thread or child thread group is created, and by the setDaemon, setMaxPriority, stop, suspend, resume, and destroy methods of class ThreadGroup.

If the thread group argument is the system thread group ( has a null parent) then this method calls checkPermission with the RuntimePermission("modifyThreadGroup") permission. If the thread group argument is *not* the system thread group, this method just returns silently.

Applications that want a stricter policy should override this method. If this method is overridden, the method that overrides it should additionally check to see if the calling thread has the RuntimePermission("modifyThreadGroup") permission, and if so, return silently. This is to ensure that code granted that permission (such as the JDK itself) is allowed to manipulate any thread.

If this method is overridden, then super.checkAccess should be called by the first statement in the overridden method, or the equivalent security check should be placed in the overridden method.

**Parameters:**g - the thread group to be checked. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to modify the thread group. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the thread group argument is null.**See Also:**[destroy](http://docs.google.com/java/lang/ThreadGroup.html#destroy()), [resume](http://docs.google.com/java/lang/ThreadGroup.html#resume()), [setDaemon](http://docs.google.com/java/lang/ThreadGroup.html#setDaemon(boolean)), [setMaxPriority](http://docs.google.com/java/lang/ThreadGroup.html#setMaxPriority(int)), [stop](http://docs.google.com/java/lang/ThreadGroup.html#stop()), [suspend](http://docs.google.com/java/lang/ThreadGroup.html#suspend()), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkExit

public void **checkExit**(int status)

Throws a SecurityException if the calling thread is not allowed to cause the Java Virtual Machine to halt with the specified status code.

This method is invoked for the current security manager by the exit method of class Runtime. A status of 0 indicates success; other values indicate various errors.

This method calls checkPermission with the RuntimePermission("exitVM."+status) permission.

If you override this method, then you should make a call to super.checkExit at the point the overridden method would normally throw an exception.

**Parameters:**status - the exit status. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to halt the Java Virtual Machine with the specified status.**See Also:**[exit](http://docs.google.com/java/lang/Runtime.html#exit(int)), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkExec

public void **checkExec**([String](http://docs.google.com/java/lang/String.html) cmd)

Throws a SecurityException if the calling thread is not allowed to create a subprocess.

This method is invoked for the current security manager by the exec methods of class Runtime.

This method calls checkPermission with the FilePermission(cmd,"execute") permission if cmd is an absolute path, otherwise it calls checkPermission with FilePermission("<<ALL FILES>>","execute").

If you override this method, then you should make a call to super.checkExec at the point the overridden method would normally throw an exception.

**Parameters:**cmd - the specified system command. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to create a subprocess. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the cmd argument is null.**See Also:**[Runtime.exec(java.lang.String)](http://docs.google.com/java/lang/Runtime.html#exec(java.lang.String)), [Runtime.exec(java.lang.String, java.lang.String[])](http://docs.google.com/java/lang/Runtime.html#exec(java.lang.String,%20java.lang.String%5B%5D)), [Runtime.exec(java.lang.String[])](http://docs.google.com/java/lang/Runtime.html#exec(java.lang.String%5B%5D)), [Runtime.exec(java.lang.String[], java.lang.String[])](http://docs.google.com/java/lang/Runtime.html#exec(java.lang.String%5B%5D,%20java.lang.String%5B%5D)), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkLink

public void **checkLink**([String](http://docs.google.com/java/lang/String.html) lib)

Throws a SecurityException if the calling thread is not allowed to dynamic link the library code specified by the string argument file. The argument is either a simple library name or a complete filename.

This method is invoked for the current security manager by methods load and loadLibrary of class Runtime.

This method calls checkPermission with the RuntimePermission("loadLibrary."+lib) permission.

If you override this method, then you should make a call to super.checkLink at the point the overridden method would normally throw an exception.

**Parameters:**lib - the name of the library. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to dynamically link the library. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the lib argument is null.**See Also:**[Runtime.load(java.lang.String)](http://docs.google.com/java/lang/Runtime.html#load(java.lang.String)), [Runtime.loadLibrary(java.lang.String)](http://docs.google.com/java/lang/Runtime.html#loadLibrary(java.lang.String)), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkRead

public void **checkRead**([FileDescriptor](http://docs.google.com/java/io/FileDescriptor.html) fd)

Throws a SecurityException if the calling thread is not allowed to read from the specified file descriptor.

This method calls checkPermission with the RuntimePermission("readFileDescriptor") permission.

If you override this method, then you should make a call to super.checkRead at the point the overridden method would normally throw an exception.

**Parameters:**fd - the system-dependent file descriptor. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to access the specified file descriptor. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the file descriptor argument is null.**See Also:**[FileDescriptor](http://docs.google.com/java/io/FileDescriptor.html), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkRead

public void **checkRead**([String](http://docs.google.com/java/lang/String.html) file)

Throws a SecurityException if the calling thread is not allowed to read the file specified by the string argument.

This method calls checkPermission with the FilePermission(file,"read") permission.

If you override this method, then you should make a call to super.checkRead at the point the overridden method would normally throw an exception.

**Parameters:**file - the system-dependent file name. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to access the specified file. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the file argument is null.**See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkRead

public void **checkRead**([String](http://docs.google.com/java/lang/String.html) file,  
 [Object](http://docs.google.com/java/lang/Object.html) context)

Throws a SecurityException if the specified security context is not allowed to read the file specified by the string argument. The context must be a security context returned by a previous call to getSecurityContext.

If context is an instance of AccessControlContext then the AccessControlContext.checkPermission method will be invoked with the FilePermission(file,"read") permission.

If context is not an instance of AccessControlContext then a SecurityException is thrown.

If you override this method, then you should make a call to super.checkRead at the point the overridden method would normally throw an exception.

**Parameters:**file - the system-dependent filename.context - a system-dependent security context. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the specified security context is not an instance of AccessControlContext (e.g., is null), or does not have permission to read the specified file. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the file argument is null.**See Also:**[getSecurityContext()](http://docs.google.com/java/lang/SecurityManager.html#getSecurityContext()), [AccessControlContext.checkPermission(java.security.Permission)](http://docs.google.com/java/security/AccessControlContext.html#checkPermission(java.security.Permission))

### checkWrite

public void **checkWrite**([FileDescriptor](http://docs.google.com/java/io/FileDescriptor.html) fd)

Throws a SecurityException if the calling thread is not allowed to write to the specified file descriptor.

This method calls checkPermission with the RuntimePermission("writeFileDescriptor") permission.

If you override this method, then you should make a call to super.checkWrite at the point the overridden method would normally throw an exception.

**Parameters:**fd - the system-dependent file descriptor. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to access the specified file descriptor. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the file descriptor argument is null.**See Also:**[FileDescriptor](http://docs.google.com/java/io/FileDescriptor.html), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkWrite

public void **checkWrite**([String](http://docs.google.com/java/lang/String.html) file)

Throws a SecurityException if the calling thread is not allowed to write to the file specified by the string argument.

This method calls checkPermission with the FilePermission(file,"write") permission.

If you override this method, then you should make a call to super.checkWrite at the point the overridden method would normally throw an exception.

**Parameters:**file - the system-dependent filename. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to access the specified file. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the file argument is null.**See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkDelete

public void **checkDelete**([String](http://docs.google.com/java/lang/String.html) file)

Throws a SecurityException if the calling thread is not allowed to delete the specified file.

This method is invoked for the current security manager by the delete method of class File.

This method calls checkPermission with the FilePermission(file,"delete") permission.

If you override this method, then you should make a call to super.checkDelete at the point the overridden method would normally throw an exception.

**Parameters:**file - the system-dependent filename. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to delete the file. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the file argument is null.**See Also:**[File.delete()](http://docs.google.com/java/io/File.html#delete()), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkConnect

public void **checkConnect**([String](http://docs.google.com/java/lang/String.html) host,  
 int port)

Throws a SecurityException if the calling thread is not allowed to open a socket connection to the specified host and port number.

A port number of -1 indicates that the calling method is attempting to determine the IP address of the specified host name.

This method calls checkPermission with the SocketPermission(host+":"+port,"connect") permission if the port is not equal to -1. If the port is equal to -1, then it calls checkPermission with the SocketPermission(host,"resolve") permission.

If you override this method, then you should make a call to super.checkConnect at the point the overridden method would normally throw an exception.

**Parameters:**host - the host name port to connect to.port - the protocol port to connect to. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to open a socket connection to the specified host and port. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the host argument is null.**See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkConnect

public void **checkConnect**([String](http://docs.google.com/java/lang/String.html) host,  
 int port,  
 [Object](http://docs.google.com/java/lang/Object.html) context)

Throws a SecurityException if the specified security context is not allowed to open a socket connection to the specified host and port number.

A port number of -1 indicates that the calling method is attempting to determine the IP address of the specified host name.

If context is not an instance of AccessControlContext then a SecurityException is thrown.

Otherwise, the port number is checked. If it is not equal to -1, the context's checkPermission method is called with a SocketPermission(host+":"+port,"connect") permission. If the port is equal to -1, then the context's checkPermission method is called with a SocketPermission(host,"resolve") permission.

If you override this method, then you should make a call to super.checkConnect at the point the overridden method would normally throw an exception.

**Parameters:**host - the host name port to connect to.port - the protocol port to connect to.context - a system-dependent security context. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the specified security context is not an instance of AccessControlContext (e.g., is null), or does not have permission to open a socket connection to the specified host and port. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the host argument is null.**See Also:**[getSecurityContext()](http://docs.google.com/java/lang/SecurityManager.html#getSecurityContext()), [AccessControlContext.checkPermission(java.security.Permission)](http://docs.google.com/java/security/AccessControlContext.html#checkPermission(java.security.Permission))

### checkListen

public void **checkListen**(int port)

Throws a SecurityException if the calling thread is not allowed to wait for a connection request on the specified local port number.

If port is not 0, this method calls checkPermission with the SocketPermission("localhost:"+port,"listen"). If port is zero, this method calls checkPermission with SocketPermission("localhost:1024-","listen").

If you override this method, then you should make a call to super.checkListen at the point the overridden method would normally throw an exception.

**Parameters:**port - the local port. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to listen on the specified port.**See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkAccept

public void **checkAccept**([String](http://docs.google.com/java/lang/String.html) host,  
 int port)

Throws a SecurityException if the calling thread is not permitted to accept a socket connection from the specified host and port number.

This method is invoked for the current security manager by the accept method of class ServerSocket.

This method calls checkPermission with the SocketPermission(host+":"+port,"accept") permission.

If you override this method, then you should make a call to super.checkAccept at the point the overridden method would normally throw an exception.

**Parameters:**host - the host name of the socket connection.port - the port number of the socket connection. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to accept the connection. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the host argument is null.**See Also:**[ServerSocket.accept()](http://docs.google.com/java/net/ServerSocket.html#accept()), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkMulticast

public void **checkMulticast**([InetAddress](http://docs.google.com/java/net/InetAddress.html) maddr)

Throws a SecurityException if the calling thread is not allowed to use (join/leave/send/receive) IP multicast.

This method calls checkPermission with the java.net.SocketPermission(maddr.getHostAddress(), "accept,connect") permission.

If you override this method, then you should make a call to super.checkMulticast at the point the overridden method would normally throw an exception.

**Parameters:**maddr - Internet group address to be used. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread is not allowed to use (join/leave/send/receive) IP multicast. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the address argument is null.**Since:** JDK1.1 **See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkMulticast

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public void **checkMulticast**([InetAddress](http://docs.google.com/java/net/InetAddress.html) maddr,  
 byte ttl)

**Deprecated.** *Use #checkPermission(java.security.Permission) instead*

Throws a SecurityException if the calling thread is not allowed to use (join/leave/send/receive) IP multicast.

This method calls checkPermission with the java.net.SocketPermission(maddr.getHostAddress(), "accept,connect") permission.

If you override this method, then you should make a call to super.checkMulticast at the point the overridden method would normally throw an exception.

**Parameters:**maddr - Internet group address to be used.ttl - value in use, if it is multicast send. Note: this particular implementation does not use the ttl parameter. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread is not allowed to use (join/leave/send/receive) IP multicast. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the address argument is null.**Since:** JDK1.1 **See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkPropertiesAccess

public void **checkPropertiesAccess**()

Throws a SecurityException if the calling thread is not allowed to access or modify the system properties.

This method is used by the getProperties and setProperties methods of class System.

This method calls checkPermission with the PropertyPermission("\*", "read,write") permission.

If you override this method, then you should make a call to super.checkPropertiesAccess at the point the overridden method would normally throw an exception.

**Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to access or modify the system properties.**See Also:**[System.getProperties()](http://docs.google.com/java/lang/System.html#getProperties()), [System.setProperties(java.util.Properties)](http://docs.google.com/java/lang/System.html#setProperties(java.util.Properties)), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkPropertyAccess

public void **checkPropertyAccess**([String](http://docs.google.com/java/lang/String.html) key)

Throws a SecurityException if the calling thread is not allowed to access the system property with the specified key name.

This method is used by the getProperty method of class System.

This method calls checkPermission with the PropertyPermission(key, "read") permission.

If you override this method, then you should make a call to super.checkPropertyAccess at the point the overridden method would normally throw an exception.

**Parameters:**key - a system property key. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to access the specified system property. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the key argument is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if key is empty.**See Also:**[System.getProperty(java.lang.String)](http://docs.google.com/java/lang/System.html#getProperty(java.lang.String)), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkTopLevelWindow

public boolean **checkTopLevelWindow**([Object](http://docs.google.com/java/lang/Object.html) window)

Returns false if the calling thread is not trusted to bring up the top-level window indicated by the window argument. In this case, the caller can still decide to show the window, but the window should include some sort of visual warning. If the method returns true, then the window can be shown without any special restrictions.

See class Window for more information on trusted and untrusted windows.

This method calls checkPermission with the AWTPermission("showWindowWithoutWarningBanner") permission, and returns true if a SecurityException is not thrown, otherwise it returns false.

If you override this method, then you should make a call to super.checkTopLevelWindow at the point the overridden method would normally return false, and the value of super.checkTopLevelWindow should be returned.

**Parameters:**window - the new window that is being created. **Returns:**true if the calling thread is trusted to put up top-level windows; false otherwise. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the window argument is null.**See Also:**[Window](http://docs.google.com/java/awt/Window.html), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkPrintJobAccess

public void **checkPrintJobAccess**()

Throws a SecurityException if the calling thread is not allowed to initiate a print job request.

This method calls checkPermission with the RuntimePermission("queuePrintJob") permission.

If you override this method, then you should make a call to super.checkPrintJobAccess at the point the overridden method would normally throw an exception.

**Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to initiate a print job request.**Since:** JDK1.1 **See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkSystemClipboardAccess

public void **checkSystemClipboardAccess**()

Throws a SecurityException if the calling thread is not allowed to access the system clipboard.

This method calls checkPermission with the AWTPermission("accessClipboard") permission.

If you override this method, then you should make a call to super.checkSystemClipboardAccess at the point the overridden method would normally throw an exception.

**Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to access the system clipboard.**Since:** JDK1.1 **See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkAwtEventQueueAccess

public void **checkAwtEventQueueAccess**()

Throws a SecurityException if the calling thread is not allowed to access the AWT event queue.

This method calls checkPermission with the AWTPermission("accessEventQueue") permission.

If you override this method, then you should make a call to super.checkAwtEventQueueAccess at the point the overridden method would normally throw an exception.

**Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to access the AWT event queue.**Since:** JDK1.1 **See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkPackageAccess

public void **checkPackageAccess**([String](http://docs.google.com/java/lang/String.html) pkg)

Throws a SecurityException if the calling thread is not allowed to access the package specified by the argument.

This method is used by the loadClass method of class loaders.

This method first gets a list of restricted packages by obtaining a comma-separated list from a call to java.security.Security.getProperty("package.access"), and checks to see if pkg starts with or equals any of the restricted packages. If it does, then checkPermission gets called with the RuntimePermission("accessClassInPackage."+pkg) permission.

If this method is overridden, then super.checkPackageAccess should be called as the first line in the overridden method.

**Parameters:**pkg - the package name. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to access the specified package. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the package name argument is null.**See Also:**[loadClass](http://docs.google.com/java/lang/ClassLoader.html#loadClass(java.lang.String,%20boolean)), [getProperty](http://docs.google.com/java/security/Security.html#getProperty(java.lang.String)), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkPackageDefinition

public void **checkPackageDefinition**([String](http://docs.google.com/java/lang/String.html) pkg)

Throws a SecurityException if the calling thread is not allowed to define classes in the package specified by the argument.

This method is used by the loadClass method of some class loaders.

This method first gets a list of restricted packages by obtaining a comma-separated list from a call to java.security.Security.getProperty("package.definition"), and checks to see if pkg starts with or equals any of the restricted packages. If it does, then checkPermission gets called with the RuntimePermission("defineClassInPackage."+pkg) permission.

If this method is overridden, then super.checkPackageDefinition should be called as the first line in the overridden method.

**Parameters:**pkg - the package name. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to define classes in the specified package.**See Also:**[ClassLoader.loadClass(java.lang.String, boolean)](http://docs.google.com/java/lang/ClassLoader.html#loadClass(java.lang.String,%20boolean)), [getProperty](http://docs.google.com/java/security/Security.html#getProperty(java.lang.String)), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkSetFactory

public void **checkSetFactory**()

Throws a SecurityException if the calling thread is not allowed to set the socket factory used by ServerSocket or Socket, or the stream handler factory used by URL.

This method calls checkPermission with the RuntimePermission("setFactory") permission.

If you override this method, then you should make a call to super.checkSetFactory at the point the overridden method would normally throw an exception.

**Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission to specify a socket factory or a stream handler factory.**See Also:**[setSocketFactory](http://docs.google.com/java/net/ServerSocket.html#setSocketFactory(java.net.SocketImplFactory)), [setSocketImplFactory](http://docs.google.com/java/net/Socket.html#setSocketImplFactory(java.net.SocketImplFactory)), [setURLStreamHandlerFactory](http://docs.google.com/java/net/URL.html#setURLStreamHandlerFactory(java.net.URLStreamHandlerFactory)), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkMemberAccess

public void **checkMemberAccess**([Class](http://docs.google.com/java/lang/Class.html)<?> clazz,  
 int which)

Throws a SecurityException if the calling thread is not allowed to access members.

The default policy is to allow access to PUBLIC members, as well as access to classes that have the same class loader as the caller. In all other cases, this method calls checkPermission with the RuntimePermission("accessDeclaredMembers") permission.

If this method is overridden, then a call to super.checkMemberAccess cannot be made, as the default implementation of checkMemberAccess relies on the code being checked being at a stack depth of 4.

**Parameters:**clazz - the class that reflection is to be performed on.which - type of access, PUBLIC or DECLARED. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the caller does not have permission to access members. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the clazz argument is null.**Since:** JDK1.1 **See Also:**[Member](http://docs.google.com/java/lang/reflect/Member.html), [checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### checkSecurityAccess

public void **checkSecurityAccess**([String](http://docs.google.com/java/lang/String.html) target)

Determines whether the permission with the specified permission target name should be granted or denied.

If the requested permission is allowed, this method returns quietly. If denied, a SecurityException is raised.

This method creates a SecurityPermission object for the given permission target name and calls checkPermission with it.

See the documentation for [SecurityPermission](http://docs.google.com/java/security/SecurityPermission.html) for a list of possible permission target names.

If you override this method, then you should make a call to super.checkSecurityAccess at the point the overridden method would normally throw an exception.

**Parameters:**target - the target name of the SecurityPermission. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the calling thread does not have permission for the requested access. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if target is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if target is empty.**Since:** JDK1.1 **See Also:**[checkPermission](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission))

### getThreadGroup

public [ThreadGroup](http://docs.google.com/java/lang/ThreadGroup.html) **getThreadGroup**()

Returns the thread group into which to instantiate any new thread being created at the time this is being called. By default, it returns the thread group of the current thread. This should be overridden by a specific security manager to return the appropriate thread group.

**Returns:**ThreadGroup that new threads are instantiated into**Since:** JDK1.1 **See Also:**[ThreadGroup](http://docs.google.com/java/lang/ThreadGroup.html)

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/SecurityManager.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/lang/SecurityException.html)   [**NEXT CLASS**](http://docs.google.com/java/lang/Short.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/SecurityManager.html)    [**NO FRAMES**](http://docs.google.com/SecurityManager.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#2s8eyo1) | [METHOD](#3rdcrjn) |

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

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

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