| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/AccessController.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/security/AccessControlException.html)   [**NEXT CLASS**](http://docs.google.com/java/security/AlgorithmParameterGenerator.html) | [**FRAMES**](http://docs.google.com/index.html?java/security/AccessController.html)    [**NO FRAMES**](http://docs.google.com/AccessController.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#tyjcwt) |

## **java.security**

Class AccessController

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

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

The AccessController class is used for access control operations and decisions.

More specifically, the AccessController class is used for three purposes:

* to decide whether an access to a critical system resource is to be allowed or denied, based on the security policy currently in effect,
* to mark code as being "privileged", thus affecting subsequent access determinations, and
* to obtain a "snapshot" of the current calling context so access-control decisions from a different context can be made with respect to the saved context.

The [checkPermission](http://docs.google.com/java/security/AccessController.html#checkPermission(java.security.Permission)) method determines whether the access request indicated by a specified permission should be granted or denied. A sample call appears below. In this example, checkPermission will determine whether or not to grant "read" access to the file named "testFile" in the "/temp" directory.

FilePermission perm = new FilePermission("/temp/testFile", "read");  
 AccessController.checkPermission(perm);

If a requested access is allowed, checkPermission returns quietly. If denied, an AccessControlException is thrown. AccessControlException can also be thrown if the requested permission is of an incorrect type or contains an invalid value. Such information is given whenever possible. Suppose the current thread traversed m callers, in the order of caller 1 to caller 2 to caller m. Then caller m invoked the checkPermission method. The checkPermission method determines whether access is granted or denied based on the following algorithm:

i = m;  
   
 while (i > 0) {  
   
 if (caller i's domain does not have the permission)  
 throw AccessControlException  
   
 else if (caller i is marked as privileged) {  
 if (a context was specified in the call to doPrivileged)   
 context.checkPermission(permission)  
 return;  
 }  
 i = i - 1;  
 };  
  
 // Next, check the context inherited when  
 // the thread was created. Whenever a new thread is created, the  
 // AccessControlContext at that time is  
 // stored and associated with the new thread, as the "inherited"  
 // context.  
   
 inheritedContext.checkPermission(permission);

A caller can be marked as being "privileged" (see [doPrivileged](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction)) and below). When making access control decisions, the checkPermission method stops checking if it reaches a caller that was marked as "privileged" via a doPrivileged call without a context argument (see below for information about a context argument). If that caller's domain has the specified permission, no further checking is done and checkPermission returns quietly, indicating that the requested access is allowed. If that domain does not have the specified permission, an exception is thrown, as usual.

The normal use of the "privileged" feature is as follows. If you don't need to return a value from within the "privileged" block, do the following:

somemethod() {  
 ...normal code here...  
 AccessController.doPrivileged(new PrivilegedAction() {  
 public Object run() {  
 // privileged code goes here, for example:  
 System.loadLibrary("awt");  
 return null; // nothing to return  
 }  
 });  
 ...normal code here...  
 }

PrivilegedAction is an interface with a single method, named run, that returns an Object. The above example shows creation of an implementation of that interface; a concrete implementation of the run method is supplied. When the call to doPrivileged is made, an instance of the PrivilegedAction implementation is passed to it. The doPrivileged method calls the run method from the PrivilegedAction implementation after enabling privileges, and returns the run method's return value as the doPrivileged return value (which is ignored in this example).

If you need to return a value, you can do something like the following:

somemethod() {  
 ...normal code here...  
 String user = (String) AccessController.doPrivileged(  
 new PrivilegedAction() {  
 public Object run() {  
 return System.getProperty("user.name");  
 }  
 }  
 );  
 ...normal code here...  
 }

If the action performed in your run method could throw a "checked" exception (those listed in the throws clause of a method), then you need to use the PrivilegedExceptionAction interface instead of the PrivilegedAction interface:

somemethod() throws FileNotFoundException {  
 ...normal code here...  
 try {  
 FileInputStream fis = (FileInputStream) AccessController.doPrivileged(  
 new PrivilegedExceptionAction() {  
 public Object run() throws FileNotFoundException {  
 return new FileInputStream("someFile");  
 }  
 }  
 );  
 } catch (PrivilegedActionException e) {  
 // e.getException() should be an instance of FileNotFoundException,  
 // as only "checked" exceptions will be "wrapped" in a  
 // PrivilegedActionException.  
 throw (FileNotFoundException) e.getException();  
 }  
 ...normal code here...  
 }

Be \*very\* careful in your use of the "privileged" construct, and always remember to make the privileged code section as small as possible.

Note that checkPermission 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 [getContext](http://docs.google.com/java/security/AccessController.html#getContext()) method and AccessControlContext class are provided for this situation. The getContext method takes a "snapshot" of the current calling context, and places it in an AccessControlContext object, which it returns. A sample call is the following:

AccessControlContext acc = AccessController.getContext()

AccessControlContext itself has a checkPermission method that makes access decisions based on the context *it* encapsulates, rather than that of the current execution thread. Code within a different context can thus call that method on the previously-saved AccessControlContext object. A sample call is the following:

acc.checkPermission(permission)

There are also times where you don't know a priori which permissions to check the context against. In these cases you can use the doPrivileged method that takes a context:

somemethod() {  
 AccessController.doPrivileged(new PrivilegedAction() {  
 public Object run() {  
 // Code goes here. Any permission checks within this  
 // run method will require that the intersection of the  
 // callers protection domain and the snapshot's  
 // context have the desired permission.  
 }  
 }, acc);  
 ...normal code here...  
 }

**See Also:**[AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html)

| **Method Summary** | |
| --- | --- |
| static void | [**checkPermission**](http://docs.google.com/java/security/AccessController.html#checkPermission(java.security.Permission))([Permission](http://docs.google.com/java/security/Permission.html) perm)            Determines whether the access request indicated by the specified permission should be allowed or denied, based on the current AccessControlContext and security policy. |
| static   | <T> T | | --- | | [**doPrivileged**](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction))([PrivilegedAction](http://docs.google.com/java/security/PrivilegedAction.html)<T> action)            Performs the specified PrivilegedAction with privileges enabled. |
| static   | <T> T | | --- | | [**doPrivileged**](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction,%20java.security.AccessControlContext))([PrivilegedAction](http://docs.google.com/java/security/PrivilegedAction.html)<T> action, [AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html) context)            Performs the specified PrivilegedAction with privileges enabled and restricted by the specified AccessControlContext. |
| static   | <T> T | | --- | | [**doPrivileged**](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedExceptionAction))([PrivilegedExceptionAction](http://docs.google.com/java/security/PrivilegedExceptionAction.html)<T> action)            Performs the specified PrivilegedExceptionAction with privileges enabled. |
| static   | <T> T | | --- | | [**doPrivileged**](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedExceptionAction,%20java.security.AccessControlContext))([PrivilegedExceptionAction](http://docs.google.com/java/security/PrivilegedExceptionAction.html)<T> action, [AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html) context)            Performs the specified PrivilegedExceptionAction with privileges enabled and restricted by the specified AccessControlContext. |
| static   | <T> T | | --- | | [**doPrivilegedWithCombiner**](http://docs.google.com/java/security/AccessController.html#doPrivilegedWithCombiner(java.security.PrivilegedAction))([PrivilegedAction](http://docs.google.com/java/security/PrivilegedAction.html)<T> action)            Performs the specified PrivilegedAction with privileges enabled. |
| static   | <T> T | | --- | | [**doPrivilegedWithCombiner**](http://docs.google.com/java/security/AccessController.html#doPrivilegedWithCombiner(java.security.PrivilegedExceptionAction))([PrivilegedExceptionAction](http://docs.google.com/java/security/PrivilegedExceptionAction.html)<T> action)            Performs the specified PrivilegedExceptionAction with privileges enabled. |
| static [AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html) | [**getContext**](http://docs.google.com/java/security/AccessController.html#getContext())()            This method takes a "snapshot" of the current calling context, which includes the current Thread's inherited AccessControlContext, and places it in an AccessControlContext object. |

| **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)) |

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

### doPrivileged

public static <T> T **doPrivileged**([PrivilegedAction](http://docs.google.com/java/security/PrivilegedAction.html)<T> action)

Performs the specified PrivilegedAction with privileges enabled. The action is performed with *all* of the permissions possessed by the caller's protection domain.

If the action's run method throws an (unchecked) exception, it will propagate through this method.

Note that any DomainCombiner associated with the current AccessControlContext will be ignored while the action is performed.

**Parameters:**action - the action to be performed. **Returns:**the value returned by the action's run method. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the action is null**See Also:**[doPrivileged(PrivilegedAction,AccessControlContext)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction,%20java.security.AccessControlContext)), [doPrivileged(PrivilegedExceptionAction)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedExceptionAction)), [doPrivilegedWithCombiner(PrivilegedAction)](http://docs.google.com/java/security/AccessController.html#doPrivilegedWithCombiner(java.security.PrivilegedAction)), [DomainCombiner](http://docs.google.com/java/security/DomainCombiner.html)

### doPrivilegedWithCombiner

public static <T> T **doPrivilegedWithCombiner**([PrivilegedAction](http://docs.google.com/java/security/PrivilegedAction.html)<T> action)

Performs the specified PrivilegedAction with privileges enabled. The action is performed with *all* of the permissions possessed by the caller's protection domain.

If the action's run method throws an (unchecked) exception, it will propagate through this method.

This method preserves the current AccessControlContext's DomainCombiner (which may be null) while the action is performed.

**Parameters:**action - the action to be performed. **Returns:**the value returned by the action's run method. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the action is null**Since:** 1.6 **See Also:**[doPrivileged(PrivilegedAction)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction)), [DomainCombiner](http://docs.google.com/java/security/DomainCombiner.html)

### doPrivileged

public static <T> T **doPrivileged**([PrivilegedAction](http://docs.google.com/java/security/PrivilegedAction.html)<T> action,  
 [AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html) context)

Performs the specified PrivilegedAction with privileges enabled and restricted by the specified AccessControlContext. The action is performed with the intersection of the permissions possessed by the caller's protection domain, and those possessed by the domains represented by the specified AccessControlContext.

If the action's run method throws an (unchecked) exception, it will propagate through this method.

**Parameters:**action - the action to be performed.context - an *access control context* representing the restriction to be applied to the caller's domain's privileges before performing the specified action. If the context is null, then no additional restriction is applied. **Returns:**the value returned by the action's run method. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the action is null**See Also:**[doPrivileged(PrivilegedAction)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction)), [doPrivileged(PrivilegedExceptionAction,AccessControlContext)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedExceptionAction,%20java.security.AccessControlContext))

### doPrivileged

public static <T> T **doPrivileged**([PrivilegedExceptionAction](http://docs.google.com/java/security/PrivilegedExceptionAction.html)<T> action)  
 throws [PrivilegedActionException](http://docs.google.com/java/security/PrivilegedActionException.html)

Performs the specified PrivilegedExceptionAction with privileges enabled. The action is performed with *all* of the permissions possessed by the caller's protection domain.

If the action's run method throws an *unchecked* exception, it will propagate through this method.

Note that any DomainCombiner associated with the current AccessControlContext will be ignored while the action is performed.

**Parameters:**action - the action to be performed **Returns:**the value returned by the action's run method **Throws:** [PrivilegedActionException](http://docs.google.com/java/security/PrivilegedActionException.html) - if the specified action's run method threw a *checked* exception [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the action is null**See Also:**[doPrivileged(PrivilegedAction)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction)), [doPrivileged(PrivilegedExceptionAction,AccessControlContext)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedExceptionAction,%20java.security.AccessControlContext)), [doPrivilegedWithCombiner(PrivilegedExceptionAction)](http://docs.google.com/java/security/AccessController.html#doPrivilegedWithCombiner(java.security.PrivilegedExceptionAction)), [DomainCombiner](http://docs.google.com/java/security/DomainCombiner.html)

### doPrivilegedWithCombiner

public static <T> T **doPrivilegedWithCombiner**([PrivilegedExceptionAction](http://docs.google.com/java/security/PrivilegedExceptionAction.html)<T> action)  
 throws [PrivilegedActionException](http://docs.google.com/java/security/PrivilegedActionException.html)

Performs the specified PrivilegedExceptionAction with privileges enabled. The action is performed with *all* of the permissions possessed by the caller's protection domain.

If the action's run method throws an *unchecked* exception, it will propagate through this method.

This method preserves the current AccessControlContext's DomainCombiner (which may be null) while the action is performed.

**Parameters:**action - the action to be performed. **Returns:**the value returned by the action's run method **Throws:** [PrivilegedActionException](http://docs.google.com/java/security/PrivilegedActionException.html) - if the specified action's run method threw a *checked* exception [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the action is null**Since:** 1.6 **See Also:**[doPrivileged(PrivilegedAction)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction)), [doPrivileged(PrivilegedExceptionAction,AccessControlContext)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedExceptionAction,%20java.security.AccessControlContext)), [DomainCombiner](http://docs.google.com/java/security/DomainCombiner.html)

### doPrivileged

public static <T> T **doPrivileged**([PrivilegedExceptionAction](http://docs.google.com/java/security/PrivilegedExceptionAction.html)<T> action,  
 [AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html) context)  
 throws [PrivilegedActionException](http://docs.google.com/java/security/PrivilegedActionException.html)

Performs the specified PrivilegedExceptionAction with privileges enabled and restricted by the specified AccessControlContext. The action is performed with the intersection of the the permissions possessed by the caller's protection domain, and those possessed by the domains represented by the specified AccessControlContext.

If the action's run method throws an *unchecked* exception, it will propagate through this method.

**Parameters:**action - the action to be performedcontext - an *access control context* representing the restriction to be applied to the caller's domain's privileges before performing the specified action. If the context is null, then no additional restriction is applied. **Returns:**the value returned by the action's run method **Throws:** [PrivilegedActionException](http://docs.google.com/java/security/PrivilegedActionException.html) - if the specified action's run method threw a *checked* exception [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the action is null**See Also:**[doPrivileged(PrivilegedAction)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedAction)), [doPrivileged(PrivilegedExceptionAction,AccessControlContext)](http://docs.google.com/java/security/AccessController.html#doPrivileged(java.security.PrivilegedExceptionAction,%20java.security.AccessControlContext))

### getContext

public static [AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html) **getContext**()

This method takes a "snapshot" of the current calling context, which includes the current Thread's inherited AccessControlContext, and places it in an AccessControlContext object. This context may then be checked at a later point, possibly in another thread.

**Returns:**the AccessControlContext based on the current context.**See Also:**[AccessControlContext](http://docs.google.com/java/security/AccessControlContext.html)

### checkPermission

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

Determines whether the access request indicated by the specified permission should be allowed or denied, based on the current AccessControlContext and security policy. This method quietly returns if the access request is permitted, or throws a suitable AccessControlException otherwise.

**Parameters:**perm - the requested permission. **Throws:** [AccessControlException](http://docs.google.com/java/security/AccessControlException.html) - if the specified permission is not permitted, based on the current security policy. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified permission is null and is checked based on the security policy currently in effect.

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/AccessController.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/security/AccessControlException.html)   [**NEXT CLASS**](http://docs.google.com/java/security/AlgorithmParameterGenerator.html) | [**FRAMES**](http://docs.google.com/index.html?java/security/AccessController.html)    [**NO FRAMES**](http://docs.google.com/AccessController.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#tyjcwt) |

[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).