java.security

Class Permission

java.lang.Object
    java.security.Permission

All Implemented Interfaces:

Serializable, Guard

Direct Known Subclasses:

AllPermission, BasicPermission, FilePermission, MBeanPermission, PrivateCredentialPermission, ServicePermission, SocketPermission, UnresolvedPermission

---

```
public abstract class Permission
extends Object
implements Guard, Serializable
```

Abstract class for representing access to a system resource. All permissions have a name (whose interpretation depends on the subclass), as well as abstract functions for defining the semantics of the particular Permission subclass.

Most Permission objects also include an "actions" list that tells the actions that are permitted for the object. For example, for a `java.io.FilePermission` object, the permission name is the pathname of a file (or directory), and the actions list (such as "read, write") specifies which actions are granted for the specified file (or for files in the specified directory). The actions list is optional for Permission objects, such as `java.lang.RuntimePermission`, that don't need such a list; you either have the named permission (such as "system.exit") or you don't.

An important method that must be implemented by each subclass is the `implies` method to compare Permissions. Basically, "permission p1 implies permission p2" means that if one is granted permission p1, one is naturally granted permission p2. Thus, this is not an equality test, but rather more of a subset test.

Permission objects are similar to String objects in that they are immutable once they have been created. Subclasses should not provide methods that can change the state of a permission once it has been created.

See Also:

`Permissions`, `PermissionCollection`, Serialized Form

Constructor Summary

Constructors

Constructor and Description
`Permission(String name)` Constructs a permission with the specified name.

Method Summary

Methods

Modifier and Type	Method and Description
void	`checkGuard(Object object)`

	Implements the guard interface for a permission.
abstract boolean	**equals**(**Object** obj) Checks two Permission objects for equality.
abstract **String**	**getActions**() Returns the actions as a String.
String	**getName**() Returns the name of this Permission.
abstract int	**hashCode**() Returns the hash code value for this Permission object.
abstract boolean	**implies**(**Permission** permission) Checks if the specified permission's actions are "implied by" this object's actions.
PermissionCollection	**newPermissionCollection**() Returns an empty PermissionCollection for a given Permission object, or null if one is not defined.
String	**toString**() Returns a string describing this Permission.

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

Permission

```
public Permission(String name)
```

Constructs a permission with the specified name.

Parameters:

name - name of the Permission object being created.

Method Detail

checkGuard

```
public void checkGuard(Object object)
                throws SecurityException
```

Implements the guard interface for a permission. The SecurityManager.checkPermission method is called, passing this permission object as the permission to check. Returns silently if access is granted. Otherwise, throws a SecurityException.

Specified by:

checkGuard in interface Guard

Parameters:

object - the object being guarded (currently ignored).

Throws:

SecurityException - if a security manager exists and its checkPermission method doesn't allow access.

See Also:

`Guard`, `GuardedObject`, `SecurityManager.checkPermission(java.security.Permission)`

implies

```
public abstract boolean implies(Permission permission)
```

Checks if the specified permission's actions are "implied by" this object's actions.

This must be implemented by subclasses of Permission, as they are the only ones that can impose semantics on a Permission object.

The `implies` method is used by the AccessController to determine whether or not a requested permission is implied by another permission that is known to be valid in the current execution context.

Parameters:

`permission` - the permission to check against.

Returns:

true if the specified permission is implied by this object, false if not.

equals

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

Checks two Permission objects for equality.

Do not use the `equals` method for making access control decisions; use the `implies` method.

Overrides:

`equals` in class `Object`

Parameters:

`obj` - the object we are testing for equality with this object.

Returns:

true if both Permission objects are equivalent.

See Also:

`Object.hashCode()`, `HashMap`

hashCode

```
public abstract int hashCode()
```

Returns the hash code value for this Permission object.

The required `hashCode` behavior for Permission Objects is the following:

- Whenever it is invoked on the same Permission object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two Permission objects are equal according to the `equals` method, then calling the `hashCode` method on each of the two Permission objects must produce the same integer result.

Overrides:

`hashCode` in class `Object`

Returns:

a hash code value for this object.

See Also:

`Object.equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

getName

```
public final String getName()
```

Returns the name of this Permission. For example, in the case of a `java.io.FilePermission`, the name will be a pathname.

Returns:

the name of this Permission.

getActions

```
public abstract String getActions()
```

Returns the actions as a String. This is abstract so subclasses can defer creating a String representation until one is needed. Subclasses should always return actions in what they consider to be their canonical form. For example, two FilePermission objects created via the following:

```
perm1 = new FilePermission(p1,"read,write");
perm2 = new FilePermission(p2,"write,read");
```

both return "read,write" when the `getActions` method is invoked.

Returns:

the actions of this Permission.

newPermissionCollection

```
public PermissionCollection newPermissionCollection()
```

Returns an empty PermissionCollection for a given Permission object, or null if one is not defined. Subclasses of class Permission should override this if they need to store their permissions in a particular PermissionCollection object in order to provide the correct semantics when the `PermissionCollection.implies` method is called. If null is returned, then the caller of this method is free to store permissions of this type in any PermissionCollection they choose (one that uses a Hashtable, one that uses a Vector, etc).

Returns:

a new PermissionCollection object for this type of Permission, or null if one is not defined.

toString

```
public String toString()
```

Returns a string describing this Permission. The convention is to specify the class name, the permission name, and the actions in the following format: '("ClassName" "name" "actions")', or '("ClassName" "name")' if actions list is null or empty.

Overrides:

`toString` in class `Object`

Returns:

information about this Permission.