Table of Contents

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Chapter 19. javax.security.auth and Subpackages

This chapter documents the `javax.security.auth` package and its subpackages, which, together, form the Java Authentication and Authorization Service, or JAAS. Before being integrated into Java 1.4, JAAS was available as a standard extension, which is why these packages have the "javax" prefix. The individual packages are the following:

`javax.security.auth`

This top-level package defines the `Subject` class that is central to JAAS.

`javax.security.auth.callback`

This package defines a callback API to enable communication (such as the exchange of a username and password) between a low-level login module and the end-user.

`javax.security.auth.kerberos`

This package contains JAAS classes related to the Kerberos network authentication protocol.

`javax.security.auth.login`

This package defines the `LoginContext` class and related classes used by applications to perform a JAAS login.

`javax.security.auth.spi`

This package defines the "service provider interface" for JAAS.

`javax.security.auth.x500`

This package includes JAAS classes related to X.500 principals.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Package javax.security.auth

Java 1.4

This is the top-level package of the Java Authentication and Authorization Service (JAAS). The key class is `Subject`, which represents an authenticated user, and defines static methods that allow Java code be run as (i.e., using the permissions of) a specified `Subject`. The remaining classes and interfaces in this package are important parts of the JAAS infrastructure, but are not commonly used in application code. Applications do not create `Subject` objects directly, but typically obtain them from a `javax.security.auth.login.LoginContext` constructed with a `javax.security.auth.callback.CallbackHandler`.

Interfaces

```
public interface Destroyable;
public interface Refreshable;
```

Classes

```
public final class AuthPermission extends java.security.BasicPermission;
public abstract class Policy;
public final class PrivateCredentialPermission extends java.security.Permission;
public final class Subject implements Serializable;
public class SubjectDomainCombiner implements java.security.DomainCombiner;
```

Exceptions

```
public class DestroyFailedException extends Exception;
public class RefreshFailedException extends Exception;
```

AuthPermission javax.security.auth

Java 1.4 *serializable permission*

This `java.security.Permission` class governs the use of various methods in this package and its subpackages. The target name of the permission specifies which methods are allowed; `AuthPermission` objects have no actions list. Application programmers never need to use this class directly. System implementors may need to use it, and system administrators who configure security policies may need to be familiar with the following table of target names and the permissions they represent:

Target name	Gives permission to
doAs	Invoke `Subject.doAs( )` methods.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Target name	Gives permission to
doAsPrivileged	Invoke `Subject.doAsPriviliged( )` methods.
getSubject	Invoke `Subject.getSubject( )`.
getSubjectFromDomainCombiner	Invoke `SubjectDomainCombiner.getSubject( )`.
setReadOnly	Invoke `Subject.setReadOnly( )`.
modifyPrincipals	Modify the `Set` of principals associated with a `Subject`.
modifyPublicCredentials	Modify the `Set` of public credentials associated with a `Subject`.
modifyPrivateCredentials	Modify the `Set` of private credentials associated with a `Subject`.
refreshCredential	Invoke the `refresh( )` method of a `Refreshable` credential class.
destroyCredential	Invoke the `destroy( )` method of a `Destroyable` credential class.
createLoginContext.*name*	Instantiate a `LoginContext` with the specified *name*. If *name* is * , it allows a `LoginContext` of any name to be created.
getLoginConfiguration	Invoke the `getConfiguration( )` method of `javax.security.auth.login.Configuration`.
setLoginConfiguration	Invoke the `setConfiguration( )` method of `javax.security.auth.login.Configuration`.
refreshLoginConfiguration	Invoke the `refresh( )` method of `javax.security.auth.login.Configuration`.

javax.security.auth.AuthPermission


```
public final class AuthPermission extends java.security.BasicPermission {
// Public Constructors
     public AuthPermission(String name);
     public AuthPermission(String name, String actions);
}
```

Destroyable javax.security.auth

Java 1.4

Classes that encapsulate sensitive information, such as security credentials, may implement this interface to provide an API that allows the sensitive information to be destroyed or erased. The `destroy( )` method erases or clears the sensitive information. It may throw a `DestroyFailedException` if the information cannot be erased for any reason. It may also throw a `SecurityException` if the caller does not have whatever permissions are required. Once `destroy( )` has been called on an object, the `isDestroyed( )` method returns `true`. Once an object has been destroyed, any other methods it defines may throw an `IllegalStateException`.

```
public interface Destroyable {
// Public Instance Methods
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

```
        void destroy( ) throws DestroyFailedException;
        boolean isDestroyed( );
    }
```

Implementations

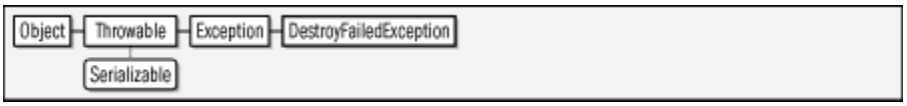
`java.security.KeyStore.PasswordProtection`,
`javax.security.auth.kerberos.KerberosKey`,
`javax.security.auth.kerberos.KerberosTicket`,
`javax.security.auth.x500.X500PrivateCredential`

DestroyFailedException — javax.security.auth

Java 1.4 — ***serializable checked***

Signals that the `destroy( )` method of a `Destroyable` object did not succeed.

Figure 19-1. javax.security.auth.DestroyFailedException




```
public class DestroyFailedException extends Exception {
// Public Constructors
     public DestroyFailedException( );
     public DestroyFailedException(String msg);
}
```

Thrown By

`java.security.KeyStore.PasswordProtection.destroy( )`,
`Destroyable.destroy( )`,
`javax.security.auth.kerberos.KerberosKey.destroy( )`,
`javax.security.auth.kerberos.KerberosTicket.destroy( )`

Policy — javax.security.auth

Java 1.4; Deprecated in 1.4 — ***@Deprecated***

This deprecated class represents a Subject-based security policy. Because the JAAS API (this package and its subpackages) were introduced as an extension to the core Java platform, this class was required to augment the `java.security.Policy` class which, prior to Java 1.4, had no provisions for Subject-based authorization. In Java 1.4, however,

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

`java.security.Policy` has been extended to represent security policies based on code origin, code signers, and subjects. Thus, this class is no longer required and has been deprecated.

```
public abstract class Policy {
// Protected Constructors
     protected Policy( );
// Public Class Methods
     public static javax.security.auth.Policy getPolicy( );
     public static void setPolicy(javax.security.auth.Policy policy);
// Public Instance Methods
     public abstract java.security.PermissionCollection getPermissions(Subject subject,
        java.security.CodeSource cs);
     public abstract void refresh( );
}
```

PrivateCredentialPermission javax.security.auth

Java 1.4 ***serializable permission***

This `Permission` class protects access to private credential objects belonging to a `Subject` (as specified by a set of one or more `Principal` objects). Application programmers rarely need to use it. System programmers implementing new private credentials classes may need to use it, and system administrators configuring security policy files should be familiar with it.

The only defined action for `PrivateCredentialPermssion` is "read". The target name for this permission has a complex syntax and specifies the name of the credential class and a list of one or more principals. Each principal is specified as the name of the `Principal` class followed by the principal name in quotes. For example, a security policy file might contain a statement like the following to allow permission to read the private `KerberosKey` credentials of a `KerberosPrincipal` named "david".

```
permission javax.security.auth.PrivateCredentialPermission
    "javax.security.auth.kerberos.KerberosKey \
            javax.security.auth.kerberos.KerberosPrincipal \"david\"",
    "read";
```

The target name syntax for `PrivateCredentialPermission` also allows the use of the "*" wildcard in place of the credential class name or in place of the `Principal` class name and/or name.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Figure 19-2. javax.security.auth.PrivateCredentialPermission


```
public final class PrivateCredentialPermission extends java.security.Permission {
// Public Constructors
     public PrivateCredentialPermission(String name, String actions);
// Public Instance Methods
     public String getCredentialClass( );
     public String[ ][ ] getPrincipals( );
// Public Methods Overriding Permission
     public boolean equals(Object obj);
     public String getActions( );
     public int hashCode( );
     public boolean implies(java.security.Permission p);
     public java.security.PermissionCollection newPermissionCollection( );     constant
}
```

Refreshable javax.security.auth

Java 1.4

A class implements this interface if its instances that have a limited period of validity (as some security credentials do) and need to be periodically "refreshed" in order to remain valid. `isCurrent( )` returns `true` if the object is currently valid, and `false` if it has expired and needs to be refreshed. `refresh( )` attempts to revalidate or extend the validity of the object. It throws a `RefreshFailedException` if it does not succeed. (And may also throw a `SecurityException` if the caller does not have the requisite permissions.)

```
public interface Refreshable {
// Public Instance Methods
     boolean isCurrent( );
     void refresh( ) throws RefreshFailedException;
}
```

Implementations

`javax.security.auth.kerberos.KerberosTicket`

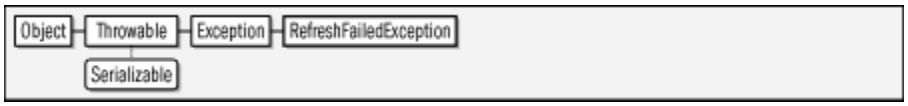
RefreshFailedException javax.security.auth

Java 1.4 ***serializable checked***

Signals that the `refresh( )` method of a `Refreshable` object failed.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Figure 19-3. javax.security.auth.RefreshFailedException




```
public class RefreshFailedException extends Exception {
// Public Constructors
     public RefreshFailedException( );
     public RefreshFailedException(String msg);
}
```

Thrown By

`Refreshable.refresh( )`, `javax.security.auth.kerberos.KerberosTicket.refresh( )`

Subject — javax.security.auth

Java 1.4 — ***serializable***

The `Subject` class is the key abstraction of the JAAS API. It represents a person or other entity, and consists of:

`Subject` defines methods that allow you to retreive each of these three sets, or to retreive a subset of each set that contains only objects of a specified `Class`. Unless the `Subject` is read-only, you can use the methods of `java.util.Set` to modify each of the three sets. Once `setReadOnly( )` has been called, however, the sets become immutable and their contents may not be modified.

Application code does not typically create `Subject` objects itself. Instead, it obtains a `Subject` that represents the authenticated user of the application by calling the `login( )` and `getSubject( )` methods of a `javax.security.auth.login.LoginContext` object.

Once an authenticated `Subject` has been obtained from a `LoginContext`, an application can call the `doAs( )` method to run code using the permissions granted to that `Subject` combined with the permissions granted to the code itself. `doAs( )` runs the code defined in the `run( )` method of a `PrivilegedAction` or `PrivilegedExceptionAction` object. `doAsPrivileged( )` is a similar method but executes the specified `run( )` method using the Subject's permissions only, unconstrained by unprivileged code in the call stack.

Note that many of the methods of this class throw a `SecurityException` if the caller has not been granted the requisite `AuthPermission`.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Figure 19-4. javax.security.auth.Subject


```
public final class Subject implements Serializable {
// Public Constructors
     public Subject( );
     public Subject(boolean readOnly, java.util.Set<? extends java.security.Principal>
        principals, java.util.Set<?> pubCredentials,
        java.util.Set<?> privCredentials);
// Public Class Methods
     public static Object doAs(Subject subject, java.security.PrivilegedExceptionAction
        action) throws java.security.PrivilegedActionException;
     public static Object doAs(Subject subject, java.security.PrivilegedAction action);
     public static Object doAsPrivileged(Subject subject, java.security.
        PrivilegedExceptionAction action, java.security.AccessControlContext acc)
        throws java.security.PrivilegedActionException;
     public static Object doAsPrivileged(Subject subject, java.security.PrivilegedAction
        action, java.security.AccessControlContext acc);
     public static Subject getSubject(java.security.AccessControlContext acc);
// Public Instance Methods
     public java.util.Set<java.security.Principal> getPrincipals( );
     public <T extends java.security.Principal> java.util.Set<T> getPrincipals(Class<T> c);
     public java.util.Set<Object> getPrivateCredentials( );
     public <T> java.util.Set<T> getPrivateCredentials(Class<T> c);
     public java.util.Set<Object> getPublicCredentials( );
     public <T> java.util.Set<T> getPublicCredentials(Class<T> c);
     public boolean isReadOnly( );                                    default:false
     public void setReadOnly( );
// Public Methods Overriding Object
     public boolean equals(Object o);
     public int hashCode( );
     public String toString( );
}
```

Passed To

`java.security.AuthProvider.login( )`,
`javax.security.auth.Policy.getPermissions( )`,
`SubjectDomainCombiner.SubjectDomainCombiner( )`,
`javax.security.auth.login.LoginContext.LoginContext( )`,
`javax.security.auth.spi.LoginModule.initialize( )`

Returned By

`SubjectDomainCombiner.getSubject( )`,
`javax.security.auth.login.LoginContext.getSubject( )`

SubjectDomainCombiner javax.security.auth

Java 1.4

This class implements the `DomainCombiner` interface. It is used to merge permissions based on code source and code signers with permissions granted to the specified

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

`Subject`. A `SubjectDomainCombiner` is created by the `Subject.doAs( )` and `Subject.doAsPrivileged( )` methods for use in by the `AccessControlContext`.

Figure 19-5. javax.security.auth.SubjectDomainCombiner


```
public class SubjectDomainCombiner implements java.security.DomainCombiner {
// Public Constructors
     public SubjectDomainCombiner(Subject subject);
// Public Instance Methods
     public Subject getSubject( );
// Methods Implementing DomainCombiner
     public java.security.ProtectionDomain[ ] combine(java.security.ProtectionDomain[ ]
        currentDomains,
        java.security.ProtectionDomain[ ] assignedDomains);
}
```

Package javax.security.auth.callback

Java 1.4

This package defines a mechanism that allows the low-level code of a `javax.security.auth.spi.LoginModule` to interact with the end-user of an application to obtain a username, password, or other authentication-related information. The `LoginModule` sends messages and requests for information in the form of objects that implement the `Callback` interface. An application that wants to authenticate a user provides (via a `javax.security.auth.login.LoginContext`) a `CallbackHandler` object to convert these `Callback` objects into text or GUI-based interactions with the user. An application that want to provide a customized login interface must implement its own `CallbackHandler`. The `CallbackHandler` API consists of only a single method, but the implementation of that method can require a substantial amount of code. See the various `Callback` classes for directions on how a `CallbackHandler` should handle them.

Sun's J2SE SDK for Java 1.4 ships with two implementations of `CallbackHandler`, both in the package `com.sun.security.auth.callback`. Although these classes are not guaranteed to exist in all distributions, text-based applications may use the `TextCallbackHandler`, and GUI-based applications may use the `DialogCallbackHandler`. Programmers wanting to write a custom `CallbackHandler` may also find it useful to study the source code of these two existing handlers.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Interfaces

```
public interface Callback;
public interface CallbackHandler;
```

Classes

```
public class ChoiceCallback implements Callback, Serializable;
public class ConfirmationCallback implements Callback, Serializable;
public class LanguageCallback implements Callback, Serializable;
public class NameCallback implements Callback, Serializable;
public class PasswordCallback implements Callback, Serializable;
public class TextInputCallback implements Callback, Serializable;
public class TextOutputCallback implements Callback, Serializable;
```

Exceptions

```
public class UnsupportedCallbackException extends Exception;
```

Callback javax.security.auth.callback

Java 1.4

This interface defines no methods but serves as a "marker interface" to identify the type of objects that can be passed to the `handle( )` method of a `CallbackHandler`. All of the classes in this package, with the exception of `UnsupportedCallbackException` implement this interface.

```
public interface Callback {
}
```

Implementations

`ChoiceCallback`, `ConfirmationCallback`, `LanguageCallback`, `NameCallback`, `PasswordCallback`, `TextInputCallback`, `TextOutputCallback`

Passed To

`CallbackHandler.handle( )`, `UnsupportedCallbackException.UnsupportedCallbackException( )`

Returned By

`UnsupportedCallbackException.getCallback( )`

CallbackHandler javax.security.auth.callback

Java 1.4

A `CallbackHandler` is responsible for communication between the end-user of an application and the `javax.security.auth.spi.LoginModule` that is performing authentication of that user on behalf of the

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

`javax.security.auth.login.LoginContext` instantiated by the application. When an application needs to authenticate a user, it creates a `LoginContext` and specifies a `CallbackHandler` object for that context. The underlying `LoginModule` uses the `CallbackHandler` to communicate with the end user—for example prompting them to enter a name and password.

The `LoginModule` passes an array of objects that implement the `Callback` interface to the `handle( )` method of `CallbackHandler`. The `handle( )` method must determine the type of `Callback` object, and display the information and/or prompt for the input it represents. Different `Callback` classes have different purposes and must be handled differently. `NameCallback` and `PasswordCallback` are two of the most commonly used: they represent requests for the user's name and password. `TextOutputCallback` is also common: it represents a request to display a message (such as "Authentication Failed") to the user. See the descriptions of the individual `Callback` classes for information on how a CallbackHandler should handle them. `CallbackHandler` implementations are not required to support every type of Callback and my throw an `UnsupportedCallbackException` if passed a `Callback` object of a type they do not recognize or do not support.

The `handle( )` method is passed an array of `Callback` objects. A `CallbackHandler` (such as a typical console-based handler) may choose to handle the `Callback` objects one at a time, prompting for and returning the user's input before moving on to the next. Or (for example in GUI-based handlers) it may choose to present all of the callbacks in a single unified "login dialog box". `LoginModule` implementations may, of course, call the `handle( )` method more than once. Note, finally, that if a `CallbackHandler` implementation has knowledge of the user from some other source, it is allowed to handle certain callbacks automatically, such as automatically providing the user's name for a `NameCallback`.

Java installations may have a default `CallbackHandler` registered by setting the `auth.login.defaultCallbackHandler` security property to the name of the implementing class. No such default is defined by the default security policy that ships with Sun's distribution of Java 1.4. Sun's Java 1.4 SDK does include `CallbackHandler` implementations to perform text-based and GUI-based communication in the classes `TextCallbackHandler` and `DialogCallbackHandler` in the `com.sun.security.auth.callback` package. Note that these are part of Sun's implementation, and are not part of the specification; they are not guaranteed to exist in all releases.

```
public interface CallbackHandler {
// Public Instance Methods
     void handle(Callback[ ] callbacks)
        throws java.io.IOException, UnsupportedCallbackException;
}
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Passed To

`java.security.AuthProvider.{login( ),setCallbackHandler( )}`,
`java.security.KeyStore.CallbackHandlerProtection.CallbackHandlerProtection( )`,
`javax.security.auth.login.LoginContext.LoginContext( )`,
`javax.security.auth.spi.LoginModule.initialize( )`

Returned By

`java.security.KeyStore.CallbackHandlerProtection.getCallbackHandler( )`

ChoiceCallback javax.security.auth.callback

Java 1.4 ***serializable***

A Callback of this type represents a request to display set of text choices and allow the user to select one or more of them. A `CallbackHandler`, should display the prompt returned by `getPrompt( )` and also the strings returned by `getChoices( )`. If `allowMultipleSelections( )` is `true`, then it should allow the user to select zero or more; otherwise, it should only allow the user to select a single one. In either case, the `CallbackHandler` should also call `getDefaultChoice( )` and make the choice at the returned index the default choice. When the user has made her selection, the `CallbackHandler` should pass the index of a single selection to `setSelectedIndex( )`, or the indexes of multiple selections to `setSelectedIndexes( )`.

Figure 19-6. javax.security.auth.callback.ChoiceCallback


```
public class ChoiceCallback implements Callback, Serializable {
// Public Constructors
     public ChoiceCallback(String prompt, String[ ] choices, int defaultChoice,
        boolean multipleSelectionsAllowed);
// Public Instance Methods
     public boolean allowMultipleSelections( );
     public String[ ] getChoices( );
     public int getDefaultChoice( );
     public String getPrompt( );
     public int[ ] getSelectedIndexes( );
     public void setSelectedIndex(int selection);
     public void setSelectedIndexes(int[ ] selections);
}
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

ConfirmationCallback **javax.security.auth.callback**

Java 1.4 ***serializable***

A `Callback` of this type represents a request to ask the user a yes/no or multiple-choice question. A `CallbackHandler` should first call `getPrompt( )` to obtain the text of the question. It should also call `getMessageType( )` to determine the message type (`INFORMATION`, `WARNING`, or `ERROR`) and present the question to the user in a suitable manner based on that type.

Next, the `CallbackHandler` must determine the appropriate set of responses to the question. It does this by calling `getOptionType( )`. The return values have the following meanings:

In each of these cases, the `CallbackHandler` should also call `getDefaultOption( )` to determine which response should be presented as the default response. If `getOptionType( )` returned `UNSPECIFIED_TYPE`, then `getDefaultOption( )` returns an index into the array of options returned by `getOptions( )`. Otherwise `getDefaultOption( )` returns one of the constants `YES`, `NO`, `OK`, or `CANCEL`.

When the user has selected a response to the callback, the `CallbackHandler` should pass that response to `setSelectedIndex( )`. The response value should be one of the constants `YES`, `NO`, `OK`, or `CANCEL`, or an index into the array of options returned by `getOptions( )`.

Figure 19-7. javax.security.auth.callback.ConfirmationCallback


```
public class ConfirmationCallback implements Callback, Serializable {
// Public Constructors
     public ConfirmationCallback(int messageType, String[ ] options, int defaultOption);
     public ConfirmationCallback(int messageType, int optionType, int defaultOption);
     public ConfirmationCallback(String prompt, int messageType, String[ ] options,
        int defaultOption);
     public ConfirmationCallback(String prompt, int messageType, int optionType,
        int defaultOption);
// Public Constants
     public static final int CANCEL;                                    =2
     public static final int ERROR;                                     =2
     public static final int INFORMATION;                               =0
     public static final int NO;                                        =1
     public static final int OK;                                        =3
     public static final int OK_CANCEL_OPTION;                          =2
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

```
     public static final int UNSPECIFIED_OPTION;                    =-1
     public static final int WARNING;                               =1
     public static final int YES;                                   =0
     public static final int YES_NO_CANCEL_OPTION;                  =1
     public static final int YES_NO_OPTION;                         =0
// Public Instance Methods
     public int getDefaultOption( );
     public int getMessageType( );
     public String[ ] getOptions( );
     public int getOptionType( );
     public String getPrompt( );
     public int getSelectedIndex( );
     public void setSelectedIndex(int selection);
}
```

LanguageCallback javax.security.auth.callback

Java 1.4 ***serializable***

This `Callback` class represents a request for the user's preferred language (as represented by a `Locale` object), which a `LoginModule` can use to localize things such as prompts and error messages in subsequent `Callback` objects. If a `CallbackHandler` already has knowledge of the user's preferred langauge, it is not required to prompt the user for this information and can simply pass an appropriate `Locale` object to `setLocale( )`.

Figure 19-8. javax.security.auth.callback.LanguageCallback


```
public class LanguageCallback implements Callback, Serializable {
// Public Constructors
     public LanguageCallback( );
// Public Instance Methods
     public java.util.Locale getLocale( );                          default:null
     public void setLocale(java.util.Locale locale);
}
```

NameCallback javax.security.auth.callback

Java 1.4 ***serializable***

This `Callback` class represents a request for the username or other text that identifies the user to be authenticated. An interactive `CallbackHandler` should call

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

`getPrompt( )` and `getDefaultName( )` and should display the returned prompt and optionally, the returned default name to the user. When the user has entered a name (or accepted the default name) the handler should pass the user's input to `setName( )`.

Figure 19-9. javax.security.auth.callback.NameCallback


```
public class NameCallback implements Callback, Serializable {
// Public Constructors
     public NameCallback(String prompt);
     public NameCallback(String prompt, String defaultName);
// Public Instance Methods
     public String getDefaultName( );
     public String getName( );
     public String getPrompt( );
     public void setName(String name);
}
```

PasswordCallback javax.security.auth.callback

Java 1.4 ***serializable***

This `Callback` class represents a request for a password. A `CallbackHandler` should handle it by displaying the prompt returned by `getPrompt( )` and then allowing the user the enter a password. When the user has entered the password, it should pass the entered text to `setPassword( )`. If `isEchoOn( )` returns true, then the `Handler` should display the password as the user types it.

Figure 19-10. javax.security.auth.callback.PasswordCallback


```
public class PasswordCallback implements Callback, Serializable {
// Public Constructors
     public PasswordCallback(String prompt, boolean echoOn);
// Public Instance Methods
     public void clearPassword( );
     public char[ ] getPassword( );
     public String getPrompt( );
     public boolean isEchoOn( );
     public void setPassword(char[ ] password);
}
```

TextInputCallback javax.security.auth.callback

Java 1.4 ***serializable***

A `Callback` of this type is a request to prompt the user for text input; it is essentially a generic version of `NameCallback`. A `CallbackHandler` should call `getPrompt( )` and should display the returned prompt text to the user. It should then allow the user to enter text, and provide the option of selecting the default text returned by `getDefaultText( )`. When the user has entered text (or selected the default text) it should pass the user's input to `setText( )`.

Figure 19-11. javax.security.auth.callback.TextInputCallback


```
public class TextInputCallback implements Callback, Serializable {
// Public Constructors
     public TextInputCallback(String prompt);
     public TextInputCallback(String prompt, String defaultText);
// Public Instance Methods
     public String getDefaultText( );
     public String getPrompt( );
     public String getText( );
     public void setText(String text);
}
```

TextOutputCallback javax.security.auth.callback

Java 1.4 ***serializable***

A `Callback` of this type represents a request to display text to the user. A callback handler should call `getMessage( )` and display the returned string to the user. It should also call `getMessageType( )` and use the returned value (which is one of the constants defined by the class) to indicate the type or severity of the information.

Figure 19-12. javax.security.auth.callback.TextOutputCallback


```
public class TextOutputCallback implements Callback, Serializable {
// Public Constructors
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

```
    public TextOutputCallback(int messageType, String message);
// Public Constants
    public static final int ERROR;                                    =2
    public static final int INFORMATION;                              =0
    public static final int WARNING;                                  =1
// Public Instance Methods
    public String getMessage( );
    public int getMessageType( );
}
```

UnsupportedCallbackException javax.security.auth.callback

Java 1.4 ***serializable checked***

`CallbackHandler` implementations may throw exceptions of this type from their `handle( )` method if a `Callback` object passed to that method is of an unrecognized or unsupported type. Note that the offending `Callback` object must be passed to the constructor method.

Figure 19-13. javax.security.auth.callback.UnsupportedCallbackException


```
public class UnsupportedCallbackException extends Exception {
// Public Constructors
    public UnsupportedCallbackException(Callback callback);
    public UnsupportedCallbackException(Callback callback, String msg);
// Public Instance Methods
    public Callback getCallback( );
}
```

Thrown By

`CallbackHandler.handle( )`

Package javax.security.auth.kerberos

Java 1.4

This package defines classes for use with Kerberos: a secure network authentication protocol. They are primarily of interest to system-level programmers writing Kerberos-based `javax.security.auth.spi.LoginModule` implementations. Developers writing Kerberos-enabled applications should use the `org.ietf.jgss` package. A full

description of Kerberos is beyond the scope of this book; so it is assumed that the reader is familar with Kerberos authentication.

Classes

```
public final class DelegationPermission extends java.security.BasicPermission
        implements Serializable;
public class KerberosKey implements javax.security.auth.Destroyable,
        javax.crypto.SecretKey;
public final class KerberosPrincipal implements java.security.Principal,
        Serializable;
public class KerberosTicket implements javax.security.auth.Destroyable,
        javax.security.auth.Refreshable, Serializable;
public final class ServicePermission extends java.security.Permission
        implements Serializable;
```

DelegationPermission javax.security.auth.kerberos

Java 1.4 ***serializable permission***

This `java.security.Permission` class governs the delegation of Kerberos tickets from a Kerberos principal to a Kerberos service for use on behalf of the original principal. The target name of a `DelegationPermission` consists of the principal names of two Kerberos services. The first specifies the service that is being delegated to, and the second specifies the service that is to be used by the first on behalf of the original Kerberos principal.

Figure 19-14. javax.security.auth.kerberos.DelegationPermission


```
public final class DelegationPermission extends java.security.BasicPermission
        implements Serializable {
// Public Constructors
     public DelegationPermission(String principals);
     public DelegationPermission(String principals, String actions);
// Public Methods Overriding BasicPermission
     public boolean equals(Object obj);
     public int hashCode( );
     public boolean implies(java.security.Permission p);
     public java.security.PermissionCollection newPermissionCollection( );
}
```

KerberosKey javax.security.auth.kerberos

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Java 1.4 ***serializable***

This class is a `javax.crypto.SecretKey` implementation that represents the secret key of a Kerberos principal. A Kerberos-based `javax.security.auth.spi.LoginModule` implementation instantiates a `KerberosKey` object and stores it in the private credential set of the authenticated `Subject` it creates.

Figure 19-15. javax.security.auth.kerberos.KerberosKey


```
public class KerberosKey implements javax.security.auth.Destroyable,
        javax.crypto.SecretKey {
// Public Constructors
     public KerberosKey(KerberosPrincipal principal, char[ ] password,
        String algorithm);
     public KerberosKey(KerberosPrincipal principal, byte[ ] keyBytes, int keyType,
        int versionNum);
// Public Instance Methods
     public final int getKeyType( );
     public final KerberosPrincipal getPrincipal( );
     public final int getVersionNumber( );
// Methods Implementing Destroyable
     public void destroy( ) throws javax.security.auth.DestroyFailedException;
     public boolean isDestroyed( );
// Methods Implementing Key
     public final String getAlgorithm( );
     public final byte[ ] getEncoded( );
     public final String getFormat( );
// Public Methods Overriding Object
     public String toString( );
}
```

KerberosPrincipal javax.security.auth.kerberos

Java 1.4 ***serializable***

This class represents a Kerberos principal, specified as a principal name with an optional realm. If no realm is specified in the name, the default realm (from the *krb5.conf* configuration file or from the `java.security.krb5.realm` system property) is used.

Figure 19-16. javax.security.auth.kerberos.KerberosPrincipal

```
public final class KerberosPrincipal implements java.security.Principal, Serializable {
// Public Constructors
     public KerberosPrincipal(String name);
     public KerberosPrincipal(String name, int nameType);
// Public Constants
     public static final int KRB_NT_PRINCIPAL;                          =1
     public static final int KRB_NT_SRV_HST;                            =3
     public static final int KRB_NT_SRV_INST;                           =2
     public static final int KRB_NT_SRV_XHST;                           =4
     public static final int KRB_NT_UID;                                =5
     public static final int KRB_NT_UNKNOWN;                            =0
// Public Instance Methods
     public int getNameType( );
     public String getRealm( );
// Methods Implementing Principal
     public boolean equals(Object other);
     public String getName( );
     public int hashCode( );
     public String toString( );
}
```

Passed To

`KerberosKey.KerberosKey( ),KerberosTicket.KerberosTicket( )`

Returned By

`KerberosKey.getPrincipal( ),KerberosTicket.{getClient( ), getServer( )}`

KerberosTicket javax.security.auth.kerberos

Java 1.4 ***serializable***

This class represents a Kerberos ticket: a credential used to authenticate a Kerberos principal to some Kerberos-enabled network service. A Kerberos-based `javax.security.auth.spi.LoginModule` implementation will instantiate a `KerberosTicket` object and store it in the private credential set of the authenticated `Subject` it creates.

Figure 19-17. javax.security.auth.kerberos.KerberosTicket


```
public class KerberosTicket implements javax.security.auth.Destroyable,
        javax.security.auth.Refreshable, Serializable {
// Public Constructors
     public KerberosTicket(byte[ ] asn1Encoding, KerberosPrincipal client,
        KerberosPrincipal server, byte[ ] sessionKey,
        int keyType, boolean[ ] flags,
        java.util.Date authTime, java.util.Date startTime,
        java.util.Date endTime, java.util.Date renewTill,
        java.net.InetAddress[ ] clientAddresses);
// Public Instance Methods
     public final java.util.Date getAuthTime( );
     public final KerberosPrincipal getClient( );
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

```
    public final java.net.InetAddress[ ] getClientAddresses( );
    public final byte[ ] getEncoded( );
    public final java.util.Date getEndTime( );
    public final boolean[ ] getFlags( );
    public final java.util.Date getRenewTill( );
    public final KerberosPrincipal getServer( );
    public final javax.crypto.SecretKey getSessionKey( );
    public final int getSessionKeyType( );
    public final java.util.Date getStartTime( );
    public final boolean isForwardable( );
    public final boolean isForwarded( );
    public final boolean isInitial( );
    public final boolean isPostdated( );
    public final boolean isProxiable( );
    public final boolean isProxy( );
    public final boolean isRenewable( );
// Methods Implementing Destroyable
    public void destroy( ) throws javax.security.auth.DestroyFailedException;
    public boolean isDestroyed( );
// Methods Implementing Refreshable
    public boolean isCurrent( );
    public void refresh( ) throws javax.security.auth.RefreshFailedException;
// Public Methods Overriding Object
    public String toString( );
}
```

ServicePermission javax.security.auth.kerberos

Java 1.4 ***serializable permission***

This `java.security.Permission` class protects access to the Kerberos tickets used to access a specified service. The target name of of a `ServicePermission` is the Kerberos principal name of the service. The action for the `ServicePermission` is either "initiate" for clients or "accept" for servers.

Figure 19-18. javax.security.auth.kerberos.ServicePermission


```
public final class ServicePermission extends java.security.Permission implements Serializable {
// Public Constructors
    public ServicePermission(String servicePrincipal, String action);
// Public Methods Overriding Permission
    public boolean equals(Object obj);
    public String getActions( );
    public int hashCode( );
    public boolean implies(java.security.Permission p);
    public java.security.PermissionCollection newPermissionCollection( );
}
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Package javax.security.auth.login

Java 1.4

This package defines the `LoginContext` class which is one of the primary JAAS classes used by application programmers. To authenticate a user, an application creates a `LoginContext` object, specifying the application name (used to lookup the type of authentication required for that application in the `Configuration`) and usually specifying a `javax.security.auth.callback.CallbackHandler` for communication between the user and the underlying login modules. Next, the application calls the `login( )` method of the `LoginContext` to perform the actual login. If this method returns without throwing a `LoginException`, then the user was sucessfully authenticated, and the `getSubject( )` method of `LoginContext` returns a `javax.security.auth.Subject` representing the user. The code might look like this:

```
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
// Get a default GUI-based CallbackHandler
CallbackHandler h = new com.sun.security.auth.callback.DialogCallbackHandler( );
// Try to create a LoginContext for use with this application
LoginContext context;
try {
  context = new LoginContext("MyAppName", h);
}
catch(LoginException e) {
  System.err.println("LoginContext configuration error: " + e.getMessage( ));
  System.exit(-1);
}
// Now use that context to authenticate the user
try {
  context.login( );
}
catch(LoginException e) {
  System.err.println("Authentication failed: " + e.getMessage( ));
  System.exit(-1);  // Or we could allow them to try again.
}
// If we get here, authentication was successful, so get the Subject that
// represents the authenticated user.
Subject subject = context.getSubject( );
```

In order to make this kind of authentication work correctly, a fair bit of configuration is required in various files in the `jre/lib/security` directory of the Java installation and possibly elsewhere. In particular, a login configuration file is required to specify which login modules are required to authenticate users for a particular application (some applications may require more than one). A description of how to do this is beyond the scope of this reference. See the `Configuration` class for a run-time representation of the login configuration information, however.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Classes

```
public class AppConfigurationEntry;
public static class AppConfigurationEntry.LoginModuleControlFlag;
public abstract class Configuration;
public class LoginContext;
```

Exceptions

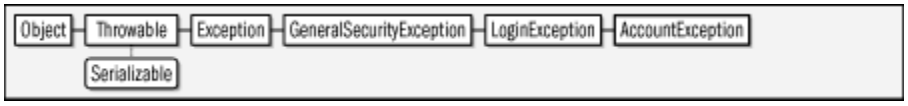
```
public class LoginException extends java.security.GeneralSecurityException;
   public class AccountException extends LoginException;
      public class AccountExpiredException extends AccountException;
      public class AccountLockedException extends AccountException;
      public class AccountNotFoundException extends AccountException;
   public class CredentialException extends LoginException;
      public class CredentialExpiredException extends CredentialException;
      public class CredentialNotFoundException extends CredentialException;
   public class FailedLoginException extends LoginException;
```

AccountException javax.security.auth.login

Java 5.0 ***serializable checked***

A `LoginException` exception of this type signals a problem logging in to the specified account. Subclasses provide more detail.

Figure 19-19. javax.security.auth.login.AccountException




```
public class AccountException extends LoginException {
// Public Constructors
     public AccountException( );
     public AccountException(String msg);
}
```

Subclasses

`AccountExpiredException`, `AccountLockedException`, `AccountNotFoundException`

AccountExpiredException javax.security.auth.login

Java 1.4 ***serializable checked***

Signals that login failed because the user's account has expired. Prior to Java 5.0, this exception was a direct subclass of `LoginException`.

Figure 19-20. javax.security.auth.login.AccountExpiredException

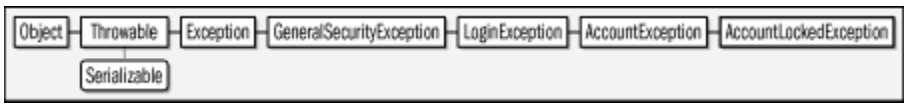

```
public class AccountExpiredException extends AccountException {
// Public Constructors
     public AccountExpiredException( );
     public AccountExpiredException(String msg);
}
```

AccountLockedException javax.security.auth.login

Java 5.0 ***serializable checked***

An exception of this type indicates that the account for which login was attempted has been "locked" or otherwise made unavailable. See also `AccountExpiredException`.

Figure 19-21. javax.security.auth.login.AccountLockedException



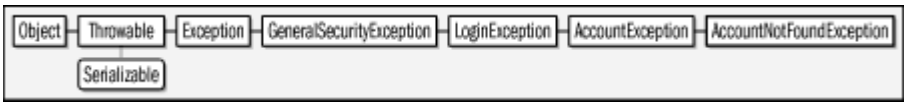

```
public class AccountLockedException extends AccountException {
// Public Constructors
     public AccountLockedException( );
     public AccountLockedException(String msg);
}
```

AccountNotFoundException javax.security.auth.login

Java 5.0 ***serializable checked***

An exception of this type indicates that the account specified in a login attempt does not exist.

Figure 19-22. javax.security.auth.login.AccountNotFoundException




```
public class AccountNotFoundException extends AccountException {
// Public Constructors
```

```
     public AccountNotFoundException( );
     public AccountNotFoundException(String msg);
}
```

AppConfigurationEntry javax.security.auth.login

Java 1.4

An instance of this class represents a login module to be used for user authentication for a particular application. It encapsulates three pieces of information: the class name of the `javax.security.auth.spi.LoginModule` implementation that is to be used, a "control flag" that specifies whether authentication by that module is required or optional, and a `java.util.Map` of arbitrary string name/value pairs of options for the login module.

```
public class AppConfigurationEntry {
// Public Constructors
     public AppConfigurationEntry(String loginModuleName, AppConfigurationEntry.
        LoginModuleControlFlag controlFlag, java.util.Map<String,?> options);
// Nested Types
     public static class LoginModuleControlFlag;
// Public Instance Methods
     public AppConfigurationEntry.LoginModuleControlFlag getControlFlag( );
     public String getLoginModuleName( );
     public java.util.Map<String,?> getOptions( );
}
```

Returned By

`Configuration.getAppConfigurationEntry( )`

AppConfigurationEntry.LoginModuleControlFlag javax.security.auth.login

Java 1.4

This inner class defines a "control flag" type and four specific instances of that type. The constants defined by this class specify whether a login module is required or optional, and have the following meanings:

```
public static class AppConfigurationEntry.LoginModuleControlFlag {
// No Constructor
// Public Constants
     public static final AppConfigurationEntry.LoginModuleControlFlag OPTIONAL;
     public static final AppConfigurationEntry.LoginModuleControlFlag REQUIRED;
     public static final AppConfigurationEntry.LoginModuleControlFlag REQUISITE;
     public static final AppConfigurationEntry.LoginModuleControlFlag SUFFICIENT;
// Public Methods Overriding Object
     public String toString( );
}
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Passed To

`AppConfigurationEntry.AppConfigurationEntry( )`

Returned By

`AppConfigurationEntry.getControlFlag( )`

Configuration — javax.security.auth.login

Java 1.4

This abstract class is a representation of the system and user login configuration files. The static `getConfiguration( )` method returns the global `Configuration` object, and the static `setConfiguration( )` allows that global object to be replaced with some other implementation. The instance method `refresh( )` causes a `Configuration` to re-read the underlying configuration files. `getAppConfigurationEntry( )` is the key method: it returns an array of `AppConfigurationEntry` objects that represent the set of login modules to be used for applications with the specified name. `LoginContext` uses this class to determine which login modules to use to authenticate a user of the named application. Application programmers do not typically need to use this class themselves. See the documentation for your Java implementation for the syntax of the underlying login configuration files.

```
public abstract class Configuration {
// Protected Constructors
     protected Configuration( );
// Public Class Methods
     public static Configuration getConfiguration( );                    synchronized
     public static void setConfiguration(Configuration configuration);
// Public Instance Methods
     public abstract AppConfigurationEntry[ ] getAppConfigurationEntry(String name);
     public abstract void refresh( );
}
```

Passed To

`LoginContext.LoginContext( )`

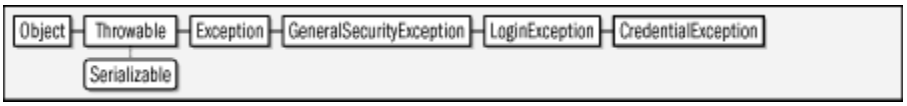
CredentialException — javax.security.auth.login

Java 5.0 — ***serializable checked***

An exception of this type indicates a problem with the credential (e.g., the password) presented during the login attempt. Subclasses provide more detail.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Figure 19-23. javax.security.auth.login.CredentialException




```
public class CredentialException extends LoginException {
// Public Constructors
     public CredentialException( );
     public CredentialException(String msg);
}
```

Subclasses

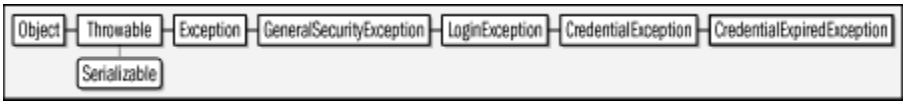
`CredentialExpiredException`, `CredentialNotFoundException`

CredentialExpiredException — javax.security.auth.login

Java 1.4 — ***serializable checked***

Signals that a login failed because a credential (such as a password) has expired and is no longer valid. Prior to Java 5.0, this is a direct subclass of `LoginException`.

Figure 19-24. javax.security.auth.login.CredentialExpiredException




```
public class CredentialExpiredException extends CredentialException {
// Public Constructors
     public CredentialExpiredException( );
     public CredentialExpiredException(String msg);
}
```

CredentialNotFoundException — javax.security.auth.login

Java 5.0 — ***serializable checked***

An exception of this type indicates that a credential (such as a Kerberos ticket) necessary for login could not be found. This is not the same as presenting an invalid credential, which results in a `FailedLoginException`.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Figure 19-25. javax.security.auth.login.CredentialNotFoundException

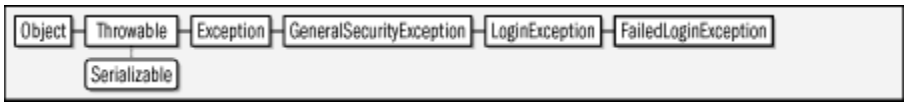

```
public class CredentialNotFoundException extends CredentialException {
// Public Constructors
     public CredentialNotFoundException( );
     public CredentialNotFoundException(String msg);
}
```

FailedLoginException javax.security.auth.login

Java 1.4 ***serializable checked***

Signals that login failed. Typically this is because an incorrect username, password, or other information was presented. Login modules that throw this exception may provide human-readable details through the `getMessage( )` method.

Figure 19-26. javax.security.auth.login.FailedLoginException




```
public class FailedLoginException extends LoginException {
// Public Constructors
     public FailedLoginException( );
     public FailedLoginException(String msg);
}
```

LoginContext javax.security.auth.login

Java 1.4

This is one of the most important classes in the JAAS API for application programmers: it defines the `login( )` method (and the corresponding `logout( )` method) that allows an application to authenticate a user. Create a `LoginContext` object using one of the public constructors. The constructor expects to be passed the name of the application, and, optionally, the `javax.security.auth.Subject` that is to be authenticated and a `javax.security.auth.callback.CallbackHandler` that is to be used for communication between the underlying login module (or modules) and the user. If no

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

`Subject` is specified, then the `LoginContext` will instantiate a new one to represent the authenticated user. If a `Subject` is supplied, then the `LoginContext` adds new entries to its sets of principals and credentials. If no `CallbackHandler` is specified, then the LoginContext attempts to instantiate one using the class name specified by the `auth.login.defaultCallbackHandler` property in the system's security properties file.

Once a `LoginContext` is successfully created, you can authenticate a user simply by calling the `login( )` method, and then calling `getSubject( )` to obtain the `Subject` object that represents the authenticated user. When this `Subject` is no longer required, you can log them out by calling the `logout( )` method.

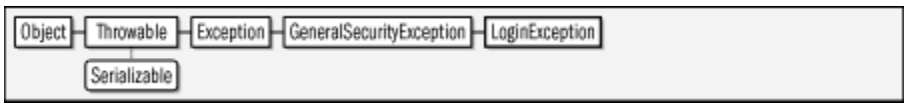
```
public class LoginContext {
// Public Constructors
     public LoginContext(String name) throws LoginException;
     public LoginContext(String name, javax.security.auth.Subject subject)
        throws LoginException;
     public LoginContext(String name, javax.security.auth.callback.
        CallbackHandler callbackHandler) throws LoginException;
     public LoginContext(String name, javax.security.auth.Subject subject,
        javax.security.auth.callback.CallbackHandler callbackHandler)
        throws LoginException;
5.0  public LoginContext(String name, javax.security.auth.Subject subject,
        javax.security.auth.callback.CallbackHandler callbackHandler,
        Configuration config) throws LoginException;
// Public Instance Methods
     public javax.security.auth.Subject getSubject( );
     public void login( ) throws LoginException;
     public void logout( ) throws LoginException;
}
```

LoginException — javax.security.auth.login

Java 1.4 — ***serializable checked***

Signals that something went wrong while creating a `LoginContext` or during the login or logout process. The subclasses of this class represent more specific exception types.

Figure 19-27. javax.security.auth.login.LoginException




```
public class LoginException extends java.security.GeneralSecurityException {
// Public Constructors
     public LoginException( );
     public LoginException(String msg);
}
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Subclasses

`AccountException`, `CredentialException`, `FailedLoginException`

Thrown By

`java.security.AuthProvider.{login( ),logout( )}`, `LoginContext.{login( ),LoginContext( ),logout( )}`, `javax.security.auth.spi.LoginModule.{abort( ),commit( ),login( ),logout( )}`

Package javax.security.auth.spi

Java 1.4

This package defines the "service provider interface" for JAAS: it defines a single `LoginModule` interface that must be implemented by developers of login modules.

Interfaces

```
public interface LoginModule;
```

LoginModule — javax.security.auth.spi

Java 1.4

Developers of login modules to be used with the JAAS authentication API must implement this interface. Because this interface is not typically used by application developers, its methods are not documented here.

```
public interface LoginModule {
// Public Instance Methods
     boolean abort( ) throws javax.security.auth.login.LoginException;
     boolean commit( ) throws javax.security.auth.login.LoginException;
     void initialize(javax.security.auth.Subject subject, javax.security.
        auth.callback.CallbackHandler callbackHandler, java.util.Map<String,?>
        sharedState, java.util.Map<String,?> options);
     boolean login( ) throws javax.security.auth.login.LoginException;
     boolean logout( ) throws javax.security.auth.login.LoginException;
}
```

Package javax.security.auth.x500

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

Java 1.4

This package defines classes for use with authentication schemes for on X.500 principals. Instances of these classes are designed to be stored in the principals and private credentials sets of `Subject` objects, and although application programmers may occasionally find the `X500Principal` class useful, they are primarily of interest to system-level programmers writing X.500-based `javax.security.auth.spi.LoginModule` implementations See also the `java.security.cert` package which contains a class representing an X.509 certificate.

Classes

```
public final class X500Principal implements java.security.Principal, Serializable;
public final class X500PrivateCredential implements javax.security.auth.Destroyable;
```

X500Principal javax.security.auth.x500

Java 1.4 ***serializable***

This class implements the `java.security.Principal` interface for entities represented by X.500 distinguished names (such as "CN=David,O=davidflanagan.com,C=US"). The constructor methods can accept the distinguished name in string form or in binary encoded form. `getName( )` returns the name in string form, using the format defined by one of the three consant values. The no-argument version of `getName( )` (the one defined by the `Principal` interface) returns the distinguished name formatted as specified by RFC 2253. Finally, `getEncoded( )` returns a binary-encoded form of the name.

Figure 19-28. javax.security.auth.x500.X500Principal


```
public final class X500Principal implements java.security.Principal, Serializable {
// Public Constructors
     public X500Principal(java.io.InputStream is);
     public X500Principal(String name);
     public X500Principal(byte[ ] name);
// Public Constants
     public static final String CANONICAL;                              ="CANONICAL"
     public static final String RFC1779;                                ="RFC1779"
     public static final String RFC2253;                                ="RFC2253"
// Public Instance Methods
     public byte[ ] getEncoded( );
     public String getName(String format);
// Methods Implementing Principal
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01

```
        public boolean equals(Object o);
        public String getName( );
        public int hashCode( );
        public String toString( );
}
```

Passed To

`java.security.cert.TrustAnchor.TrustAnchor( ), java.security.cert.X509CertSelector.{setIssuer( ),setSubject( )}, java.security.cert.X509CRLSelector.addIssuer( )`

Returned By

`java.security.cert.TrustAnchor.getCA( ), java.security.cert.X509Certificate.{getIssuerX500Principal( ), getSubjectX500Principal( )},java.security.cert.X509CertSelector.{getIssuer( ),getSubject( )}, java.security.cert.X509CRL.getIssuerX500Principal( ), java.security.cert.X509CRLEntry.getCertificateIssuer( )`

X500PrivateCredential javax.security.auth.x500

Java 1.4

This class associates a `java.security.cert.X509Certificate` with a `java.security.PrivateKey` for that certificate, and, optionally, the keystore alias used to retrieve the certificate and key from a `java.security.KeyStore`. The class defines methods to retreive the certificate, key, and alias, and also implements the methods of the `javax.security.cert.Destroyable` interface.

Figure 19-29. javax.security.auth.x500.X500PrivateCredential


```
public final class X500PrivateCredential implements javax.security.auth.Destroyable {
// Public Constructors
     public X500PrivateCredential(java.security.cert.X509Certificate cert,
        java.security.PrivateKey key);
     public X500PrivateCredential(java.security.cert.X509Certificate cert,
        java.security.PrivateKey key, String alias);
// Public Instance Methods
     public String getAlias( );
     public java.security.cert.X509Certificate getCertificate( );
     public java.security.PrivateKey getPrivateKey( );
// Methods Implementing Destroyable
     public void destroy( );
     public boolean isDestroyed( );
}
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01