Table of Contents

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

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

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

Chapter 14. java.security and Subpackages

This chapter documents the `java.security` package and its subpackages. Those packages are:

`java.security`

This large packages contains much of Java's security infrastructure, including a group of classes that provide access control through policies and permissions, and another group that provides authentication-related services such as digital signatures.

`java.security.cert`

This package defines classes and interfaces for working with public key certificates, certificate revocation lists (CRLs) and, in Java 1.4 and later, certificate chains (or certificate paths). It defines classes that should work with any type of certificate, and type-specific subclasses for X.509 certificates and CRLs.

`java.security.interfaces`

This package defines interfaces for algorithm-specific types of cryptographic keys. Providers that support those algorithms must implement these interfaces.

`java.security.spec`

This package defines classes that define a transparent, portable representation of algorithm-specific objects such as cryptographic keys. Instances of these classes can be used with any security provider.

The `java.security.acl` package is part of the Java platform, but has been superseded by access-control classes in `java.security`. It is not documented here.

Package java.security

Java 1.1

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

The `java.security` package contains the classes and interfaces that implement the Java security architecture. These classes can be divided into two broad categories. First, there are classes that implement access control and prevent untrusted code from performing sensitive operations. Second, there are authentication classes that implement message digests and digital signatures and can authenticate Java classes and other objects.

The central access control class is `AccessController`; it uses the currently installed `Policy` object to decide whether a given class has `Permission` to access a given system resource. The `Permissions` and `ProtectionDomain` classes are also important pieces of the Java access control architecture.

The key classes for authentication are `MessageDigest` and `Signature`; they compute and verify cryptographic message digests and digital signatures. These classes use public-key cryptography techniques and rely on the `PublicKey` and `PrivateKey` interfaces. They also rely on an infrastructure of related classes, such as `SecureRandom` for producing cryptographic-strength pseudorandom numbers, `KeyPairGenerator` for generating pairs of public and private keys, and `KeyStore` for managing a collection of keys and certificates. (This package defines a `Certificate` interface, but it is deprecated; see the `java.security.cert` package for the preferred `Certificate` class.)

The `CodeSource` class unites the authentication classes with the access control classes. It represents the source of a Java class as a `URL` and a set of `java.security.cert.Certificate` objects that contain the digital signatures of the code. The `AccessController` and `Policy` classes look at the `CodeSource` of a class when making access control decisions.

All the cryptographic-authentication features of this package are provider-based, which means they are implemented by security provider modules that can be plugged easily into any Java 1.2 (or later) installation. Thus, in addition to defining a security API, this package also defines a service provider interface (SPI). Various classes with names that end in Spi are part of this SPI. Security provider implementations must subclass these Spi classes, but applications never need to use them. Each security provider is represented by a `Provider` class, and the `Security` class allows new providers to be dynamically installed.

The `java.security` package contains several useful utility classes. For example, `DigestInputStream` and `DigestOutputStream` make it easy to compute message digests. `GuardedObject` provides customizable access control for an individual object. `SignedObject` protects the integrity of an arbitrary Java object by attaching a digital signature, making it easy to detect any tampering with the object. Although the `java.security` package contains cryptographic classes for authentication, it does not

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

contain classes for encryption or decryption. Instead, this functionality is part of the Java Cryptography Extension or JCE which defines the `javax.crypto` package and its subpackages. The JCE is part of the core platform in Java 1.4 and later, and is available as a standard extension to Java 1.2 and Java 1.3.

Interfaces

```
public interface Certificate;
public interface DomainCombiner;
public interface Guard;
public interface Key extends Serializable;
public interface KeyStore.Entry;
public interface KeyStore.LoadStoreParameter;
public interface KeyStore.ProtectionParameter;
public interface Principal;
public interface PrivateKey extends Key;
public interface PrivilegedAction<T>;
public interface PrivilegedExceptionAction<T>;
public interface PublicKey extends Key;
```

Enumerated Types

```
public enum KeyRep.Type;
```

Collections

```
public abstract class Provider extends java.util.Properties;
   public abstract class AuthProvider extends Provider;
```

Other Classes

```
public final class AccessControlContext;
public final class AccessController;
public class AlgorithmParameterGenerator;
public abstract class AlgorithmParameterGeneratorSpi;
public class AlgorithmParameters;
public abstract class AlgorithmParametersSpi;
public final class CodeSigner implements Serializable;
public class CodeSource implements Serializable;
public class DigestInputStream extends java.io.FilterInputStream;
public class DigestOutputStream extends java.io.FilterOutputStream;
public class GuardedObject implements Serializable;
public abstract class Identity implements Principal, Serializable;
   public abstract class IdentityScope extends Identity;
   public abstract class Signer extends Identity;
public class KeyFactory;
public abstract class KeyFactorySpi;
public final class KeyPair implements Serializable;
public abstract class KeyPairGeneratorSpi;
   public abstract class KeyPairGenerator extends KeyPairGeneratorSpi;
public class KeyRep implements Serializable;
public class KeyStore;
public abstract static class KeyStore.Builder;
public static class KeyStore.CallbackHandlerProtection implements KeyStore.
   ProtectionParameter;
public static class KeyStore.PasswordProtection
   implements javax.security.auth.Destroyable, KeyStore.ProtectionParameter;
public static final class KeyStore.PrivateKeyEntry implements KeyStore.Entry;
public static final class KeyStore.SecretKeyEntry implements KeyStore.Entry;
public static final class KeyStore.TrustedCertificateEntry implements KeyStore.
   Entry;
public abstract class KeyStoreSpi;
public abstract class MessageDigestSpi;
   public abstract class MessageDigest extends MessageDigestSpi;
public abstract class Permission implements Guard, Serializable;
   public final class AllPermission extends Permission;
   public abstract class BasicPermission extends Permission implements
      Serializable;
      public final class SecurityPermission extends BasicPermission;
```


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

```
   public final class UnresolvedPermission extends Permission implements
      Serializable;
public abstract class PermissionCollection implements Serializable;
   public final class Permissions extends PermissionCollection implements
      Serializable;
public abstract class Policy;
public class ProtectionDomain;
public static class Provider.Service;
public class SecureClassLoader extends ClassLoader;
public class SecureRandom extends java.util.Random;
public abstract class SecureRandomSpi implements Serializable;
public final class Security;
public abstract class SignatureSpi;
   public abstract class Signature extends SignatureSpi;
public final class SignedObject implements Serializable;
public final class Timestamp implements Serializable;
```

Exceptions

```
public class AccessControlException extends SecurityException;
public class GeneralSecurityException extends Exception;
   public class DigestException extends GeneralSecurityException;
   public class InvalidAlgorithmParameterException extends
   GeneralSecurityException;
   public class KeyException extends GeneralSecurityException;
      public class InvalidKeyException extends KeyException;
      public class KeyManagementException extends KeyException;
   public class KeyStoreException extends GeneralSecurityException;
   public class NoSuchAlgorithmException extends GeneralSecurityException;
   public class NoSuchProviderException extends GeneralSecurityException;
   public class SignatureException extends GeneralSecurityException;
   public class UnrecoverableEntryException extends GeneralSecurityException;
   public class UnrecoverableKeyException extends GeneralSecurityException;
public class InvalidParameterException extends IllegalArgumentException;
public class PrivilegedActionException extends Exception;
public class ProviderException extends RuntimeException;
```

AccessControlContext java.security

Java 1.2

This class encapsulates the state of a call stack. The `checkPermission( )` method can make access-control decisions based on the saved state of the call stack. Access-control checks are usually performed by the `AccessController.checkPermission( )` method, which checks that the current call stack has the required permissions. Sometimes, however, it is necessary to make access-control decisions based on a previous state of the call stack. Call `AccessController.getContext( )` to create an `AccessControlContext` for a particular call stack. In Java 1.3, this class has constructors that specify a custom context in the form of an array of `ProtectionDomain` objects and that associate a `DomainCombiner` object with an existing `AccessControlContext`. This class is used only by system-level code; typical applications rarely need to use it.

```
public final class AccessControlContext {
// Public Constructors
     public AccessControlContext(ProtectionDomain[ ] context);
1.3  public AccessControlContext(AccessControlContext acc, DomainCombiner
```


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

```
        combiner);
// Public Instance Methods
     public void checkPermission(Permission perm) throws AccessControlException;
1.3  public DomainCombiner getDomainCombiner( );
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
}
```

Passed To

`AccessController.doPrivileged( )`, `javax.security.auth.Subject.{doAsPrivileged( ), getSubject( )}`

Returned By

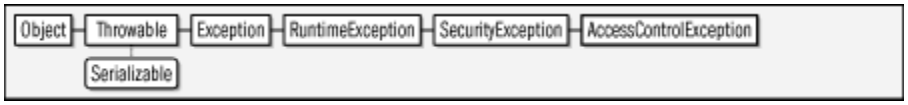
`AccessController.getContext( )`

AccessControlException java.security

Java 1.2 ***serializable unchecked***

Thrown by `AccessController` to signal that an access request has been denied. `getPermission( )` returns the `Permission` object, if any, that was involved in the denied request.

Figure 14-1. java.security.AccessControlException




```
public class AccessControlException extends SecurityException {
// Public Constructors
     public AccessControlException(String s);
     public AccessControlException(String s, Permission p);
// Public Instance Methods
     public Permission getPermission( );
}
```

Thrown By

`AccessControlContext.checkPermission( )`, `AccessController.checkPermission( )`

AccessController java.security

Java 1.2

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

The static methods of this class implement the default access-control mechanism as of Java 1.2. `checkPermission( )` traverses the call stack of the current thread and checks whether all classes in the call stack have the requested permission. If so, `checkPermission( )` returns, and the operation can proceed. If not, `checkPermission( )` throws an `AccessControlException`. As of Java 1.2, the `checkPermission( )` method of the default `java.lang.SecurityManager` calls `AccessController.checkPermission( )`. System-level code that needs to perform an access check should invoke the `SecurityManager` method rather than calling the `AccessController` method directly. Unless you are writing system-level code that must control access to system resources, you never need to use this class or the `SecurityManager.checkPermission( )` method.

The various `doPrivileged( )` methods run blocks of privileged code encapsulated in a `PrivilegedAction` or `PrivilegedExceptionAction` object. When `checkPermission( )` is traversing the call stack of a thread, it stops if it reaches a privileged block that was executed with `doPrivileged( )`. This means that privileged code can run with a full set of privileges, even if it was invoked by untrusted or lower-privileged code. See `PrivilegedAction` for more details.

The `getContext( )` method returns an `AccessControlContext` that represents the current security context of the caller. Such a context might be saved and passed to a future call (perhaps a call made from a different thread). Use the two-argument version of `doPrivileged( )` to force permission checks to check the `AccessControlContext` as well.

```
public final class AccessController {
// No Constructor
// Public Class Methods
     public static void checkPermission(Permission perm)
        throws AccessControlException;
     public static <T> T doPrivileged(PrivilegedExceptionAction<T> action)
        throws PrivilegedActionException;    naopdtive
     public static <T> T doPrivileged(PrivilegedAction<T> action);     native
     public static <T> T doPrivileged(PrivilegedExceptionAction<T> action,
        AccessControlContext context)
        throws PrivilegedActionException;     native
     public static <T> T doPrivileged(PrivilegedAction<T> action,
       AccessControlContext context);     native
     public static AccessControlContext getContext( );
}
```

AlgorithmParameterGenerator **java.security**

Java 1.2

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

This class defines a generic API for generating parameters for a cryptographic algorithm, typically a `Signature` or a `javax.crypto.Cipher`. Create an `AlgorithmParameterGenerator` by calling one of the static `getInstance( )` factory methods and specifying the name of the algorithm and, optionally, the name or `Provider` object of the desired provider. The default "SUN" provider supports the "DSA" algorithm. The "SunJCE" provider shipped with the JCE supports "DiffieHellman". Once you have obtained a generator, initialize it by calling the `init( )` method and specifying an algorithm-independent parameter size (in bits) or an algorithm-dependent `AlgorithmParameterSpec` object. You may also specify a `SecureRandom` source of randomness when you call `init( )`. Once you have created and initialized the `AlgorithmParameterGenerator`, call `generateParameters( )` to generate an `AlgorithmParameters` object.

```
public class AlgorithmParameterGenerator {
// Protected Constructors
     protected AlgorithmParameterGenerator(AlgorithmParameterGeneratorSpi
paramGenSpi, Provider provider, String algorithm);
// Public Class Methods
     public static AlgorithmParameterGenerator getInstance(String algorithm)
        throws NoSuchAlgorithmException;
1.4  public static AlgorithmParameterGenerator getInstance(String algorithm,
Provider provider) throws NoSuchAlgorithmException;
     public static AlgorithmParameterGenerator getInstance(String algorithm,
        String provider)
        throws NoSuchAlgorithmException, NoSuchProviderException;
// Public Instance Methods
     public final AlgorithmParameters generateParameters( );
     public final String getAlgorithm( );
     public final Provider getProvider( );
     public final void init(java.security.spec.AlgorithmParameterSpec
        genParamSpec) throws InvalidAlgorithmParameterException;
     public final void init(int size);
     public final void init(java.security.spec.AlgorithmParameterSpec
        genParamSpec, SecureRandom random)
        throws InvalidAlgorithmParameterException;
     public final void init(int size, SecureRandom random);
}
```

AlgorithmParameterGeneratorSpi java.security

Java 1.2

This abstract class defines the service-provider interface for algorithm-parameter generation. A security provider must implement a concrete subclass of this class for each algorithm it supports. Applications never need to use or subclass this class.

```
public abstract class AlgorithmParameterGeneratorSpi {
// Public Constructors
     public AlgorithmParameterGeneratorSpi( );
// Protected Instance Methods
     protected abstract AlgorithmParameters engineGenerateParameters( );
     protected abstract void engineInit(java.security.spec.
```


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

```
AlgorithmParameterSpec genParamSpec, SecureRandom random)
throws InvalidAlgorithmParameterException;
     protected abstract void engineInit(int size, SecureRandom random);
}
```

Passed To

`AlgorithmParameterGenerator.AlgorithmParameterGenerator( )`

AlgorithmParameters java.security

Java 1.2

This class is a generic, opaque representation of the parameters used by some cryptographic algorithm. You can create an instance of the class with one of the static `getInstance( )` factory methods, specifying the desired algorithm and, optionally, the desired provider. The default "SUN" provider supports the "DSA" algorithm. The "SunJCE" provider shipped with the JCE supports "DES", "DESede", "PBE", "Blowfish", and "DiffieHellman". Once you have obtained an `AlgorithmParameters` object, initialize it by passing an algorithm-specific `java.security.spec.AlgorithmParameterSpec` object or the encoded parameter values as a byte array to the `init( )` method. You can also create an `AlgorithmParameters` object with an `AlgorithmParameterGenerator`. `getEncoded( )` returns the initialized algorithm parameters as a byte array, using either the algorithm-specific default encoding or the named encoding format you specified.

```
public class AlgorithmParameters {
// Protected Constructors
     protected AlgorithmParameters(AlgorithmParametersSpi paramSpi, Provider
provider, String algorithm);
// Public Class Methods
     public static AlgorithmParameters getInstance(String algorithm)
throws NoSuchAlgorithmException;
     public static AlgorithmParameters getInstance(String algorithm,
String provider) throws NoSuchAlgorithmException, NoSuchProviderException;
1.4  public static AlgorithmParameters getInstance(String algorithm, Provider provider)
throws NoSuchAlgorithmException;
// Public Instance Methods
     public final String getAlgorithm( );
     public final byte[ ] getEncoded( ) throws java.io.IOException;
     public final byte[ ] getEncoded(String format) throws java.io.IOException;
     public final <T extends java.security.spec.AlgorithmParameterSpec>
T getParameterSpec(Class<T> paramSpec) throws java.security.spec.
InvalidParameterSpecException;
     public final Provider getProvider( );
     public final void init(java.security.spec.AlgorithmParameterSpec paramSpec)
throws java.security.spec.InvalidParameterSpecException;
     public final void init(byte[ ] params) throws java.io.IOException;
     public final void init(byte[ ] params, String format)
throws java.io.IOException;
// Public Methods Overriding Object
     public final String toString( );
}
```


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

Passed To

`javax.crypto.Cipher.init( ), javax.crypto.CipherSpi.engineInit( ), javax.crypto.EncryptedPrivateKeyInfo.EncryptedPrivateKeyInfo( ), javax.crypto.ExemptionMechanism.init( ), javax.crypto.ExemptionMechanismSpi.engineInit( )`

Returned By

`AlgorithmParameterGenerator.generateParameters( ), AlgorithmParameterGeneratorSpi.engineGenerateParameters( ), Signature.getParameters( ), SignatureSpi.engineGetParameters( ), javax.crypto.Cipher.getParameters( ), javax.crypto.CipherSpi.engineGetParameters( ), javax.crypto.EncryptedPrivateKeyInfo.getAlgParameters( )`

AlgorithmParametersSpi — java.security

Java 1.2

This abstract class defines the service-provider interface for `AlgorithmParameters`. A security provider must implement a concrete subclass of this class for each cryptographic algorithm it supports. Applications never need to use or subclass this class.

```
public abstract class AlgorithmParametersSpi {
// Public Constructors
     public AlgorithmParametersSpi( );
// Protected Instance Methods
     protected abstract byte[ ] engineGetEncoded( ) throws java.io.IOException;
     protected abstract byte[ ] engineGetEncoded(String format)
        throws java.io.IOException;
     protected abstract <T extends java.security.spec.AlgorithmParameterSpec>
        T engineGetParameterSpec(Class<T> paramSpec)
        throws java.security.spec.InvalidParameterSpecException;
     protected abstract void engineInit(java.security.spec.
        AlgorithmParameterSpec paramSpec)
        throws java.security.spec.InvalidParameterSpecException;
     protected abstract void engineInit(byte[ ] params)
        throws java.io.IOException;
     protected abstract void engineInit(byte[ ] params, String format)
        throws java.io.IOException;
     protected abstract String engineToString( );
}
```

Passed To

`AlgorithmParameters.AlgorithmParameters( )`

AllPermission — java.security

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

Java 1.2 ***serializable permission***

This class is a `Permission` subclass whose `implies( )` method always returns `true`. This means that code that has been granted `AllPermission` is granted all other possible permissions. This class exists to provide a convenient way to grant all permissions to completely trusted code. It should be used with care. Applications typically do not need to work directly with `Permission` objects.

Figure 14-2. java.security.AllPermission

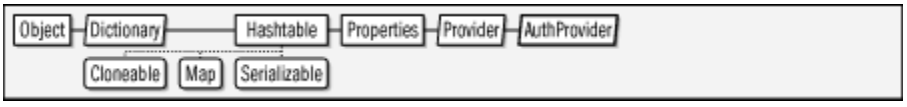

```
public final class AllPermission extends Permission {
// Public Constructors
     public AllPermission( );
     public AllPermission(String name, String actions);
// Public Methods Overriding Permission
     public boolean equals(Object obj);
     public String getActions( );      default:"<all actions>"
     public int hashCode( );           constant
     public boolean implies(Permission p);      constant
     public PermissionCollection newPermissionCollection( );
}
```

AuthProvider **java.security**

Java 5.0 ***cloneable serializable collection***

This subclass of `Provider` defines methods that allow users to "log in" before using the provider's services. An implementation of the `login( )` method should use the supplied `javax.security.auth.callback.CallbackHandler` class to request the user's password or other authentication credentials. If no callback handler is passed to `login( )`, it should use the one registered with `setCallbackHandler( )` or a default.

Figure 14-3. java.security.AuthProvider




```
public abstract class AuthProvider extends Provider {
// Protected Constructors
     protected AuthProvider(String name, double version, String info);
// Public Instance Methods
     public abstract void login(javax.security.auth.Subject subject, javax.
```

```
        security.auth.callback.CallbackHandler handler)
        throws javax.security.auth.login.LoginException;
    public abstract void logout( ) throws javax.security.auth.login.LoginException;
    public abstract void setCallbackHandler(javax.security.auth.callback.
        CallbackHandler handler);
}
```

BasicPermission java.security

Java 1.2 ***serializable permission***

This `Permission` class is the abstract superclass for a number of simple permission types. `BasicPermission` is typically subclassed to implement named permissions that have a name, or target, string, but do not support actions. The `implies( )` method of `BasicPermission` defines a simple wildcarding capability. The target "*" implies permission for any target. The target "x.*" implies permission for any target that begins with "x.". Applications typically do not need to work directly with `Permission` objects.

Figure 14-4. java.security.BasicPermission


```
public abstract class BasicPermission extends Permission
        implements Serializable {
// Public Constructors
    public BasicPermission(String name);
    public BasicPermission(String name, String actions);
// Public Methods Overriding Permission
    public boolean equals(Object obj);
    public String getActions( );
    public int hashCode( );
    public boolean implies(Permission p);
    public PermissionCollection newPermissionCollection( );
}
```

Subclasses

`java.io.SerializablePermission`, `RuntimePermission`, `java.lang.management.ManagementPermission`, `java.lang.reflect.ReflectPermission`, `java.net.NetPermission`, `SecurityPermission`, `java.util.PropertyPermission`, `java.util.logging.LoggingPermission`, `javax.net.ssl.SSLPermission`, `javax.security.auth.AuthPermission`, `javax.security.auth.kerberos.DelegationPermission`

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

Certificate — java.security

Java 1.1; Deprecated in 1.2 — ***@Deprecated***

This interface was used in Java 1.1 to represent an identity certificate. It has been deprecated as of Java 1.2 in favor of the `java.security.cert` package (see Chapter 19). See also `java.security.cert.Certificate`.

```
public interface Certificate {
// Public Instance Methods
     void decode(java.io.InputStream stream)
        throws KeyException, java.io.IOException;
     void encode(java.io.OutputStream stream)
        throws KeyException, java.io.IOException;
     String getFormat( );
     Principal getGuarantor( );
     Principal getPrincipal( );
     PublicKey getPublicKey( );
     String toString(boolean detailed);
}
```

Passed To

`Identity.{addCertificate( ),removeCertificate( )}`

Returned By

`Identity.certificates( )`

CodeSigner — java.security

Java 5.0 — ***serializable***

This class encapsulates the certificate path of a code signer and a signed timestamp. Instances are immutable. See `CodeSource` and `java.util.jar.JarEntry`.

Figure 14-5. java.security.CodeSigner


```
public final class CodeSigner implements Serializable {
// Public Constructors
     public CodeSigner(java.security.cert.CertPath signerCertPath,
        Timestamp timestamp);
// Public Instance Methods
     public java.security.cert.CertPath getSignerCertPath( );
     public Timestamp getTimestamp( );
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
     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

`CodeSource.CodeSource( )`

Returned By

`CodeSource.getCodeSigners( )`, `java.util.jar.JarEntry.getCodeSigners( )`

CodeSource — java.security

Java 1.2 — ***serializable***

This class represents the source of a Java class, as defined by the URL from which the class was loaded and the set of digital signatures attached to the class. A `CodeSource` object is created by specifying a `java.net.URL` and an array of `java.security.cert.Certificate` objects. In Java 5.0, the class has been generalized to accept an array of `CodeSigner` objects instead of `Certificate` objects. Only applications that create custom `ClassLoader` objects should ever need to use or subclass this class.

When a `CodeSource` represents a specific piece of Java code, it includes a fully qualified URL and the actual set of certificates used to sign the code. When a `CodeSource` object defines a `ProtectionDomain`, however, the URL may include wildcards, and the array of certificates is a minimum required set of signatures. The `implies( )` method of such a `CodeSource` tests whether a particular Java class comes from a matching URL and has the required set of signatures.

Figure 14-6. java.security.CodeSource


```
public class CodeSource implements Serializable {
// Public Constructors
5.0  public CodeSource(java.net.URL url, CodeSigner[ ] signers);
     public CodeSource(java.net.URL url, java.security.cert.
        Certificate[ ] certs);
// Public Instance Methods
     public final java.security.cert.Certificate[ ] getCertificates( );
5.0  public final CodeSigner[ ] getCodeSigners( );
     public final java.net.URL getLocation( );
     public boolean implies(CodeSource codesource);
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
     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

`java.net.URLClassLoader.getPermissions( ), java.security.Policy.getPermissions( ), ProtectionDomain.ProtectionDomain( ),SecureClassLoader.{defineClass( ),getPermissions( )}, javax.security.auth.Policy.getPermissions( )`

Returned By

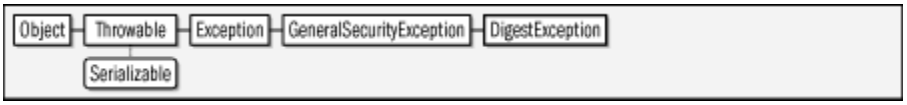
`ProtectionDomain.getCodeSource( )`

DigestException — java.security

Java 1.1 — ***serializable checked***

Signals a problem creating a message digest.

Figure 14-7. java.security.DigestException




```
public class DigestException extends GeneralSecurityException {
// Public Constructors
     public DigestException( );
5.0  public DigestException(Throwable cause);
     public DigestException(String msg);
5.0  public DigestException(String message, Throwable cause);
}
```

Thrown By

`MessageDigest.digest( ),MessageDigestSpi.engineDigest( )`

DigestInputStream — java.security

Java 1.1 — ***closeable***

This class is a byte input stream with an associated `MessageDigest` object. When bytes are read with any of the `read( )` methods, those bytes are automatically passed to the `update( )` method of the `MessageDigest`. When you have finished reading bytes, you can call the `digest( )` method of the `MessageDigest` to obtain a message digest. If you want to compute a digest just for some of the bytes read from the stream, use `on( )` to turn the digesting function on and off. Digesting is on by default; call `on(false)` to turn it off. See also `DigestOutputStream` and `MessageDigest`.

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

Figure 14-8. java.security.DigestInputStream


```
public class DigestInputStream extends java.io.FilterInputStream {
// Public Constructors
     public DigestInputStream(java.io.InputStream stream, MessageDigest digest);
// Public Instance Methods
     public MessageDigest getMessageDigest( );
     public void on(boolean on);
     public void setMessageDigest(MessageDigest digest);
// Public Methods Overriding FilterInputStream
     public int read( ) throws java.io.IOException;
     public int read(byte[ ] b, int off, int len) throws java.io.IOException;
// Public Methods Overriding Object
     public String toString( );
// Protected Instance Fields
     protected MessageDigest digest;
}
```

DigestOutputStream java.security

Java 1.1 ***closeable flushable***

This class is a byte output stream with an associated `MessageDigest` object. When bytes are written to the stream with any of the `write( )` methods, those bytes are automatically passed to the `update( )` method of the `MessageDigest`. When you have finished writing bytes, you can call the `digest( )` method of the `MessageDigest` to obtain a message digest. If you want to compute a digest just for some of the bytes written to the stream, use `on( )` to turn the digesting function on and off. Digesting is on by default; call `on(false)` to turn it off. See also `DigestInputStream` and `MessageDigest`.

Figure 14-9. java.security.DigestOutputStream


```
public class DigestOutputStream extends java.io.FilterOutputStream {
// Public Constructors
     public DigestOutputStream(java.io.OutputStream stream,
        MessageDigest digest);
// Public Instance Methods
     public MessageDigest getMessageDigest( );
     public void on(boolean on);
     public void setMessageDigest(MessageDigest digest);
// Public Methods Overriding FilterOutputStream
     public void write(int b) throws java.io.IOException;
     public void write(byte[ ] b, int off, int len) throws java.io.IOException;
// Public Methods Overriding Object
```


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

```
    public String toString( );
// Protected Instance Fields
    protected MessageDigest digest;
}
```

DomainCombiner java.security

Java 1.3

This interface defines a single `combine( )` method that combines two arrays of `ProtectionDomain` objects into a single equivalent (and perhaps optimized) array. You can associate a `DomainCombiner` with an existing `AccessControlContext` by calling the two-argument `AccessControlContext( )` constructor. Then, when the `checkPermission( )` method of the `AccessControlContext` is called or when the `AccessControlContext` is passed to a `doPrivileged( )` method of `AccessController`, the specified `DomainCombiner` merges the protection domains of the current stack frame with the protection domains encapsulated in the `AccessControlContext`. This class is used only by system-level code; typical applications rarely need to use it.

```
public interface DomainCombiner {
// Public Instance Methods
    ProtectionDomain[ ] combine(ProtectionDomain[ ] currentDomains,
       ProtectionDomain[ ] assignedDomains);
}
```

Implementations

`javax.security.auth.SubjectDomainCombiner`

Passed To

`AccessControlContext.AccessControlContext( )`

Returned By

`AccessControlContext.getDomainCombiner( )`

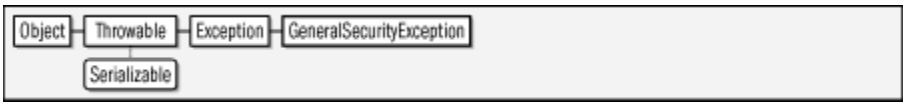
GeneralSecurityException java.security

Java 1.2 ***serializable checked***

This class is the superclass of most of the exceptions defined by the `java.security` package.

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

Figure 14-10. java.security.GeneralSecurityException




```
public class GeneralSecurityException extends Exception {
// Public Constructors
     public GeneralSecurityException( );
5.0  public GeneralSecurityException(Throwable cause);
     public GeneralSecurityException(String msg);
5.0  public GeneralSecurityException(String message, Throwable cause);
}
```

Subclasses

Too many classes to list.

Guard java.security

Java 1.2

This interface guards access to an object. The `checkGuard( )` method is passed an object to which access has been requested. If access should be granted, `checkGuard( )` should return silently. Otherwise, if access is denied, `checkGuard( )` should throw a `java.lang.SecurityException`. The `Guard` object is used primarily by the `GuardedObject` class. Note that all `Permission` objects implement the `Guard` interface.

```
public interface Guard {
// Public Instance Methods
     void checkGuard(Object object) throws SecurityException;
}
```

Implementations

`Permission`

Passed To

`GuardedObject.GuardedObject( )`

GuardedObject java.security

Java 1.2 ***serializable***

This class uses a `Guard` object to guard against unauthorized access to an arbitrary encapsulated object. Create a `GuardedObject` by specifying an object and a `Guard` for it. The `getObject( )` method calls the `checkGuard( )` method of the `Guard` to

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

determine whether access to the object should be allowed. If access is allowed, `getObject( )` returns the encapsulated object. Otherwise, it throws a `java.lang.SecurityException`.

The `Guard` object used by a `GuardedObject` is often a `Permission`. In this case, access to the guarded object is granted only if the calling code is granted the specified permission by the current security policy.

Figure 14-11. java.security.GuardedObject


```
public class GuardedObject implements Serializable {
// Public Constructors
     public GuardedObject(Object object, Guard guard);
// Public Instance Methods
     public Object getObject( ) throws SecurityException;
}
```

Identity — java.security

Java 1.1; Deprecated in 1.2 — ***@Deprecated serializable***

This deprecated class was used in Java 1.1 to represent an entity or `Principal` with an associated `PublicKey` object. In Java 1.1, the public key for a named entity could be retrieved from the system keystore with a line like the following:

```
IdentityScope.getSystemScope( ).getIdentity(name).getPublicKey( )
```

As of Java 1.2, the `Identity` class and the related `IdentityScope` and `Signer` classes have been deprecated in favor of `KeyStore` and `java.security.cert.Certificate`.

Figure 14-12. java.security.Identity


```
public abstract class Identity implements Principal, Serializable {
// Public Constructors
     public Identity(String name);
     public Identity(String name, IdentityScope scope)
        throws KeyManagementException;
// Protected Constructors
     protected Identity( );
// Public Instance Methods
```


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

```
    public void addCertificate(java.security.Certificate certificate)
       throws KeyManagementException;
    public java.security.Certificate[ ] certificates( );
    public String getInfo( );
    public PublicKey getPublicKey( );
    public final IdentityScope getScope( );
    public void removeCertificate(java.security.Certificate certificate)
       throws KeyManagementException;
    public void setInfo(String info);
    public void setPublicKey(PublicKey key) throws KeyManagementException;
    public String toString(boolean detailed);
// Methods Implementing Principal
    public final boolean equals(Object identity);
    public final String getName( );
    public int hashCode( );
    public String toString( );
// Protected Instance Methods
    protected boolean identityEquals(Identity identity);
}
```

Subclasses

`IdentityScope`, `Signer`

Passed To

`IdentityScope.{addIdentity( ), removeIdentity( )}`

Returned By

`IdentityScope.getIdentity( )`

IdentityScope java.security

Java 1.1; Deprecated in 1.2 ***@Deprecated serializable***

This deprecated class was used in Java 1.1 to represent a group of `Identity` and `Signer` objects and their associated `PublicKey` and `PrivateKey` objects. As of Java 1.2, it has been replaced by the `KeyStore` class.

Figure 14-13. java.security.IdentityScope


```
public abstract class IdentityScope extends Identity {
// Public Constructors
    public IdentityScope(String name);
    public IdentityScope(String name, IdentityScope scope)
       throws KeyManagementException;
// Protected Constructors
    protected IdentityScope( );
// Public Class Methods
    public static IdentityScope getSystemScope( );
// Protected Class Methods
    protected static void setSystemScope(IdentityScope scope);
// Public Instance Methods
    public abstract void addIdentity(Identity identity)
       throws KeyManagementException;
    public abstract Identity getIdentity(String name);
```


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

```
    public Identity getIdentity(Principal principal);
    public abstract Identity getIdentity(PublicKey key);
    public abstract java.util.Enumeration<Identity> identities( );
    public abstract void removeIdentity(Identity identity)
       throws KeyManagementException;
    public abstract int size( );
// Public Methods Overriding Identity
    public String toString( );
}
```

Passed To

`Identity.Identity( ),Signer.Signer( )`

Returned By

`Identity.getScope( )`

InvalidAlgorithmParameterException java.security

Java 1.2 ***serializable checked***

Signals that one or more algorithm parameters (usually specified by a `java.security.spec.AlgorithmParameterSpec` object) are not valid.

Figure 14-14. java.security.InvalidAlgorithmParameterException


```
public class InvalidAlgorithmParameterException
        extends GeneralSecurityException {
// Public Constructors
    public InvalidAlgorithmParameterException( );
5.0 public InvalidAlgorithmParameterException(Throwable cause);
    public InvalidAlgorithmParameterException(String msg);
5.0 public InvalidAlgorithmParameterException(String message, Throwable cause);
}
```

Thrown By

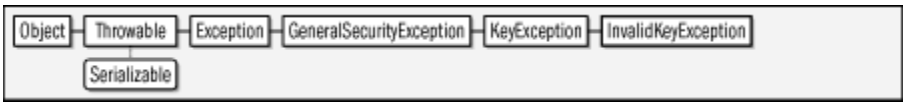
Too many methods to list.

InvalidKeyException java.security

Java 1.1 ***serializable checked***

Signals that a `Key` is not valid.

Figure 14-15. java.security.InvalidKeyException




```
public class InvalidKeyException extends KeyException {
// Public Constructors
     public InvalidKeyException( );
5.0  public InvalidKeyException(Throwable cause);
     public InvalidKeyException(String msg);
5.0  public InvalidKeyException(String message, Throwable cause);
}
```

Thrown By

Too many methods to list.

InvalidParameterException — java.security

Java 1.1 — ***serializable unchecked***

This subclass of `java.lang.IllegalArgumentException` signals that a parameter passed to a security method is not valid. This exception type is not widely used.

Figure 14-16. java.security.InvalidParameterException


```
public class InvalidParameterException extends IllegalArgumentException {
// Public Constructors
     public InvalidParameterException( );
     public InvalidParameterException(String msg);
}
```

Thrown By

`Signature.{getParameter( ),setParameter( )},SignatureSpi.{engineGetParameter( ),engineSetParameter( )}, Signer.setKeyPair( ), java.security.interfaces.DSAKeyPairGenerator.initialize( )`

Key — java.security

Java 1.1 — ***serializable***

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

This interface defines the high-level characteristics of all cryptographic keys. `getAlgorithm( )` returns the name of the cryptographic algorithm (such as RSA) used with the key. `getFormat( )` return the name of the external encoding (such as X.509) used with the key. `getEncoded( )` returns the key as an array of bytes, encoded using the format specified by `getFormat( )`.

Figure 14-17. java.security.Key


```
public interface Key extends Serializable {
// Public Constants
1.2  public static final long serialVersionUID;  =6603384152749567654
// Public Instance Methods
     String getAlgorithm( );
     byte[ ] getEncoded( );
     String getFormat( );
}
```

Implementations

`PrivateKey`, `PublicKey`, `javax.crypto.SecretKey`

Passed To

Too many methods to list.

Returned By

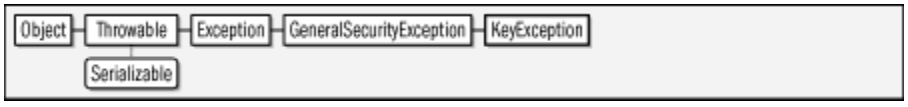
`KeyFactory.translateKey( )`, `KeyFactorySpi.engineTranslateKey( )`, `KeyStore.getKey( )`, `KeyStoreSpi.engineGetKey( )`, `javax.crypto.Cipher.unwrap( )`, `javax.crypto.CipherSpi.engineUnwrap( )`, `javax.crypto.KeyAgreement.doPhase( )`, `javax.crypto.KeyAgreementSpi.engineDoPhase( )`

KeyException java.security

Java 1.1 ***serializable checked***

Signals that something is wrong with a key. See also the subclasses `InvalidKeyException` and `KeyManagementException`.

Figure 14-18. java.security.KeyException




```
public class KeyException extends GeneralSecurityException {
// Public Constructors
```


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

```
    public KeyException( );
5.0 public KeyException(Throwable cause);
    public KeyException(String msg);
5.0 public KeyException(String message, Throwable cause);
}
```

Subclasses

`InvalidKeyException`, `KeyManagementException`

Thrown By

`java.security.Certificate.{decode( ),encode( )}`, `Signer.setKeyPair( )`

KeyFactory java.security

Java 1.2

This class translates asymmetric cryptographic keys between the two representations used by the Java Security API. `java.security.Key` is the opaque, algorithm-independent representation of a key used by most of the Security API. `java.security.spec.KeySpec` is a marker interface implemented by transparent, algorithm-specific representations of keys. `KeyFactory` is used with public and private keys; see `javax.crypto.SecretKeyFactory` if you are working with symmetric or secret keys.

To convert a `Key` to a `KeySpec` or vice versa, create a `KeyFactory` by calling one of the static `getInstance( )` factory methods specifying the name of the key algorithm (e.g., DSA or RSA) and optionally specifying the name or `Provider` object for the desired provider. Then, use `generatePublic( )` or `generatePrivate( )` to create a `PublicKey` or `PrivateKey` object from a corresponding `KeySpec`. Or use `getKeySpec( )` to obtain a `KeySpec` for a given `Key`. Because there can be more than one `KeySpec` implementation used by a particular cryptographic algorithm, you must also specify the `Class` of the `KeySpec` you desire.

If you do not need to transport keys portably between applications and/or systems, you can use a `KeyStore` to store and retrieve keys and certificates, avoiding `KeySpec` and `KeyFactory` altogether.

```
public class KeyFactory {
// Protected Constructors
    protected KeyFactory(KeyFactorySpi keyFacSpi, Provider provider,
       String algorithm);
// Public Class Methods
    public static KeyFactory getInstance(String algorithm)
       throws NoSuchAlgorithmException;
    public static KeyFactory getInstance(String algorithm, String provider)
       throws NoSuchAlgorithmException, NoSuchProviderException;
```


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

```
1.4  public static KeyFactory getInstance(String algorithm, Provider provider)
        throws NoSuchAlgorithmException;
// Public Instance Methods
     public final PrivateKey generatePrivate(java.security.spec.KeySpec keySpec)
        throws java.security.spec.InvalidKeySpecException;
     public final PublicKey generatePublic(java.security.spec.KeySpec keySpec)
        throws java.security.spec.InvalidKeySpecException;
     public final String getAlgorithm( );
     public final <T extends java.security.spec.KeySpec> T getKeySpec(Key key,
        Class<T> keySpec)
        throws java.security.spec.InvalidKeySpecException;
     public final Provider getProvider( );
     public final Key translateKey(Key key) throws InvalidKeyException;
}
```

KeyFactorySpi java.security

Java 1.2

This abstract class defines the service-provider interface for `KeyFactory`. A security provider must implement a concrete subclass of this class for each cryptographic algorithm it supports. Applications never need to use or subclass this class.

```
public abstract class KeyFactorySpi {
// Public Constructors
     public KeyFactorySpi( );
// Protected Instance Methods
     protected abstract PrivateKey engineGeneratePrivate(java.security.spec.
        KeySpec keySpec) throws java.security.spec.InvalidKeySpecException;
     protected abstract PublicKey engineGeneratePublic(java.security.spec.
        KeySpec keySpec) throws java.security.spec.InvalidKeySpecException;
     protected abstract <T extends java.security.spec.KeySpec>
        T engineGetKeySpec(Key key, Class<T> keySpec)
        throws java.security.spec.InvalidKeySpecException;
     protected abstract Key engineTranslateKey(Key key)
        throws InvalidKeyException;
}
```

Passed To

`KeyFactory.KeyFactory( )`

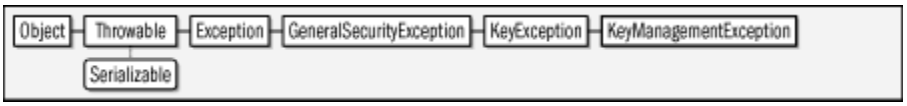
KeyManagementException java.security

Java 1.1 ***serializable checked***

Signals an exception in a key management operation. In Java 1.2, this exception is only thrown by deprecated methods.

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

Figure 14-19. java.security.KeyManagementException




```
public class KeyManagementException extends KeyException {
// Public Constructors
     public KeyManagementException( );
5.0  public KeyManagementException(Throwable cause);
     public KeyManagementException(String msg);
5.0  public KeyManagementException(String message, Throwable cause);
}
```

Thrown By

`Identity.{addCertificate( ),Identity( ),removeCertificate( ), setPublicKey( )},IdentityScope.{addIdentity( ),IdentityScope( ), removeIdentity( )},Signer.Signer( ), javax.net.ssl.SSLContext.init( ), javax.net.ssl.SSLContextSpi.engineInit( )`

KeyPair — java.security

Java 1.1 — ***serializable***

This class is a simple container for a `PublicKey` and a `PrivateKey` object. Because a `KeyPair` contains an unprotected private key, it must be used with as much caution as a `PrivateKey` object.

Figure 14-20. java.security.KeyPair


```
public final class KeyPair implements Serializable {
// Public Constructors
     public KeyPair(PublicKey publicKey, PrivateKey privateKey);
// Public Instance Methods
     public PrivateKey getPrivate( );
     public PublicKey getPublic( );
}
```

Passed To

`Signer.setKeyPair( )`

Returned By

`KeyPairGenerator.{generateKeyPair( ),genKeyPair( )}, KeyPairGeneratorSpi.generateKeyPair( )`

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

KeyPairGenerator — java.security

Java 1.1

This class generates a public/private key pair for a specified cryptographic algorithm. To create a `KeyPairGenerator`, call one of the static `getInstance( )` methods, specifying the name of the algorithm and, optionally, the name or `Provider` object of the security provider to use. The default "SUN" provider shipped with Java 1.2 supports only the "DSA" algorithm. The "SunJCE" provider of the Java Cryptography Extension (JCE) additionally supports the "DiffieHellman" algorithm.

Once you have created a `KeyPairGenerator`, initialize it by calling `initialize( )`. You can perform an algorithm-independent initialization by simply specifying the desired key size in bits. Alternatively, you can do an algorithm-dependent initialization by providing an appropriate `AlgorithmParameterSpec` object for the key-generation algorithm. In either case, you may optionally provide your own source of randomness in the guise of a `SecureRandom` object. Once you have created and initialized a `KeyPairGenerator`, call `genKeyPair( )` to create a `KeyPair` object. Remember that the `KeyPair` contains a `PrivateKey` that *must* be kept private.

For historical reasons, `KeyPairGenerator` extends `KeyPairGeneratorSpi`. Applications should not use any methods inherited from that class.

Figure 14-21. java.security.KeyPairGenerator


```
public abstract class KeyPairGenerator extends KeyPairGeneratorSpi {
// Protected Constructors
     protected KeyPairGenerator(String algorithm);
// Public Class Methods
     public static KeyPairGenerator getInstance(String algorithm)
        throws NoSuchAlgorithmException;
1.4  public static KeyPairGenerator getInstance(String algorithm,
        Provider provider) throws NoSuchAlgorithmException;
     public static KeyPairGenerator getInstance(String algorithm,
        String provider)
throws NoSuchAlgorithmException, NoSuchProviderException;
// Public Instance Methods
1.2  public final KeyPair genKeyPair( );
     public String getAlgorithm( );
1.2  public final Provider getProvider( );
1.2  public void initialize(java.security.spec.AlgorithmParameterSpec params)
        throws InvalidAlgorithmParameterException;
     public void initialize(int keysize);
// Public Methods Overriding KeyPairGeneratorSpi
     public KeyPair generateKeyPair( );          constant
1.2  public void initialize(java.security.spec.AlgorithmParameterSpec params,
        SecureRandom random)
        throws InvalidAlgorithmParameterException;     empty
```


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

```
     public void initialize(int keysize, SecureRandom random);     empty
}
```

KeyPairGeneratorSpi java.security

Java 1.2

This abstract class defines the service-provider interface for `KeyPairGenerator`. A security provider must implement a concrete subclass of this class for each cryptographic algorithm for which it can generate key pairs. Applications never need to use or subclass this class.

```
public abstract class KeyPairGeneratorSpi {
// Public Constructors
     public KeyPairGeneratorSpi( );
// Public Instance Methods
     public abstract KeyPair generateKeyPair( );
     public void initialize(java.security.spec.AlgorithmParameterSpec params,
        SecureRandom random)
        throws InvalidAlgorithmParameterException;
     public abstract void initialize(int keysize, SecureRandom random);
}
```

Subclasses

`KeyPairGenerator`

KeyRep java.security

Java 5.0 ***serializable***

This class defines a serialized representation for `Key` implementations and is typically used only by security providers, not users of the `java.security` package.

Figure 14-22. java.security.KeyRep


```
public class KeyRep implements Serializable {
// Public Constructors
     public KeyRep(KeyRep.Type type, String algorithm, String format,
        byte[ ] encoded);
// Nested Types
     public enum Type;
// Protected Instance Methods
     protected Object readResolve( ) throws java.io.ObjectStreamException;
}
```

KeyRep.Type **java.security**

Java 5.0 ***serializable comparable enum***

The constants defined by this enumerated type represent the general types of cryptographic keys: public keys, private keys, and secret keys.

```
public enum KeyRep.Type {
// Enumerated Constants
     SECRET,
     PUBLIC,
     PRIVATE;
// Public Class Methods
     public static KeyRep.Type valueOf(String name);
     public static final KeyRep.Type[ ] values( );
}
```

Passed To

`KeyRep.KeyRep( )`

KeyStore **java.security**

Java 1.2

This class represents a mapping of names, or aliases, to `Key` and `java.security.cert.Certificate` objects. Obtain a `KeyStore` object by calling one of the static `getInstance( )` methods, specifying the desired key store type and, optionally, the desired provider. Use "JKS" to specify the "Java Key Store" type defined by Sun. Because of U.S. export regulations, this default `KeyStore` supports only weak encryption of private keys. If you have the Java Cryptography Extension installed, use the type "JCEKS" and provider "SunJCE" to obtain a `KeyStore` implementation that offers much stronger password-based encryption of keys. Once you have created a `KeyStore`, use `load( )` to read its contents from a stream, supplying an optional password that verifies the integrity of the stream data. Keystores are typically read from a file named *.keystore* in the user's home directory.

The `KeyStore` API has been substantially enhanced in Java 5.0. We describe pre-5.0 methods first, and then cover Java 5.0 enhancements below. A `KeyStore` may contain both public and private key entries. A public key entry is represented by a `Certificate` object. Use `getCertificate( )` to look up a named public key certificate

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

and `setCertificateEntry( )` to add a new public key certificate to the keystore. A private key entry in the keystore contains both a password-protected `Key` and an array of `Certificate` objects that represent the certificate chain for the public key that corresponds to the private key. Use `getKey( )` and `getCertificateChain( )` to look up the key and certificate chain. Use `setKeyEntry( )` to create a new private key entry. You must provide a password when reading or writing a private key from the keystore; this password encrypts the key data, and each private key entry should have a different password. If you are using the JCE, you may also store `javax.crypto.SecretKey` objects in a `KeyStore`. Secret keys are stored like private keys, except that they do not have a certificate chain associated with them. To delete an entry from a `KeyStore`, use `deleteEntry( )`. If you modify the contents of a `KeyStore`, use `store( )` to save the keystore to a specified stream. You may specify a password that is used to validate the integrity of the data, but it is not used to encrypt the keystore.

In Java 5.0 the `KeyStore.Entry` interface defines a keystore entry. Implementations include the nested types `PrivateKeyEntry`, `SecretKeyEntry`, and `TrustedCertificateEntry`. You can get or set an entry of any type with the new methods `getEntry( )` and `setEntry( )`. These methods accept a `KeyStore.ProtectionParameter` object, such as a password represented as a `KeyStore.PasswordProtection` object. Java 5.0 also defines new `load( )` and `store( )` methods that specify a password indirectly through a `KeyStore.LoadStoreParameter`.

```
public class KeyStore {
// Protected Constructors
    protected KeyStore(KeyStoreSpi keyStoreSpi, Provider provider, String type);
// Nested Types
5.0  public abstract static class Builder;
5.0  public static class CallbackHandlerProtection
        implements KeyStore.ProtectionParameter;
5.0  public interface Entry;
5.0  public interface LoadStoreParameter;
5.0  public static class PasswordProtection
        implements javax.security.auth.Destroyable, KeyStore.ProtectionParameter;
5.0  public static final class PrivateKeyEntry
        implements KeyStore.Entry;
5.0  public interface ProtectionParameter;
5.0  public static final class SecretKeyEntry implements KeyStore.Entry;
5.0  public static final class TrustedCertificateEntry implements KeyStore.Entry;
// Public Class Methods
     public static final String getDefaultType( );
     public static KeyStore getInstance(String type) throws KeyStoreException;
     public static KeyStore getInstance(String type, String provider)
        throws KeyStoreException, NoSuchProviderException;
1.4  public static KeyStore getInstance(String type, Provider provider)
        throws KeyStoreException;
// Public Instance Methods
     public final java.util.Enumeration<String> aliases( )
        throws KeyStoreException;
     public final boolean containsAlias(String alias) throws KeyStoreException;
     public final void deleteEntry(String alias) throws KeyStoreException;
5.0  public final boolean entryInstanceOf(String alias,
        Class<? extends KeyStore.Entry> entryClass)
        throws KeyStoreException;
     public final java.security.cert.Certificate getCertificate(String alias)
```


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

```
        throws KeyStoreException;
    public final String getCertificateAlias(java.security.cert.Certificate cert)
        throws KeyStoreException;
    public final java.security.cert.Certificate[ ] getCertificateChain
        (String alias) throws KeyStoreException;
    public final java.util.Date getCreationDate(String alias)
        throws KeyStoreException;
5.0 public final KeyStore.Entry getEntry(String alias, KeyStore.
        ProtectionParameter protParam)
        throws NoSuchAlgorithmException, UnrecoverableEntryException, KeyStoreException;
    public final Key getKey(String alias, char[ ] password)
        throws KeyStoreException, NoSuchAlgorithmException, UnrecoverableKeyException;
    public final Provider getProvider( );
    public final String getType( );
    public final boolean isCertificateEntry(String alias)
        throws KeyStoreException;
    public final boolean isKeyEntry(String alias) throws KeyStoreException;
5.0 public final void load(KeyStore.LoadStoreParameter param)
        throws java.io.IOException, NoSuchAlgorithmException,
        java.security.cert.CertificateException;
    public final void load(java.io.InputStream stream, char[ ] password)
        throws java.io.IOException, NoSuchAlgorithmException,
        java.security.cert.CertificateException;
    public final void setCertificateEntry(String alias, java.security.cert.
        Certificate cert) throws KeyStoreException;
5.0 public final void setEntry(String alias, KeyStore.Entry entry,
        KeyStore.ProtectionParameter protParam)
        throws KeyStoreException;
    public final void setKeyEntry(String alias, byte[ ] key,
        java.security.cert.Certificate[ ] chain)
        throws KeyStoreException;
    public final void setKeyEntry(String alias, Key key, char[ ] password,
        java.security.cert.Certificate[ ] chain)
        throws KeyStoreException;
    public final int size( ) throws KeyStoreException;
5.0 public final void store(KeyStore.LoadStoreParameter param)
        throws KeyStoreException, java.io.IOException, NoSuchAlgorithmException,
        java.security.cert.CertificateException;
    public final void store(java.io.OutputStream stream, char[ ] password)
        throws KeyStoreException, java.io.IOException, NoSuchAlgorithmException,
        java.security.cert.CertificateException;
}
```

Passed To

`KeyStore.Builder.newInstance( ), java.security.cert.PKIXBuilderParameters.PKIXBuilderParameters( ), java.security.cert.PKIXParameters.PKIXParameters( ), javax.net.ssl.KeyManagerFactory.init( ), javax.net.ssl.KeyManagerFactorySpi.engineInit( ), javax.net.ssl.TrustManagerFactory.init( ), javax.net.ssl.TrustManagerFactorySpi.engineInit( )`

Returned By

`KeyStore.Builder.getKeyStore( )`

KeyStore.Builder — java.security

Java 5.0

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

An instance of this class encapsulates the parameters necessary to obtain a `KeyStore` object at some later time. This class is useful when you want to defer the initialization of a `KeyStore` (which may require the user to enter a password) until it is needed. See the `javax.net.ssl.KeyStoreBuilderParameters` class, for example.

```
public abstract static class KeyStore.Builder {
// Protected Constructors
     protected Builder( );
// Public Class Methods
     public static KeyStore.Builder newInstance(KeyStore keyStore,
        KeyStore.ProtectionParameter protectionParameter);
     public static KeyStore.Builder newInstance(String type, Provider provider,
        KeyStore.ProtectionParameter protection);
     public static KeyStore.Builder newInstance(String type, Provider provider,
        java.io.File file,
KeyStore.ProtectionParameter protection);
// Public Instance Methods
     public abstract KeyStore getKeyStore( ) throws KeyStoreException;
     public abstract KeyStore.ProtectionParameter getProtectionParameter
        (String alias) throws KeyStoreException;
}
```

Passed To

`javax.net.ssl.KeyStoreBuilderParameters.KeyStoreBuilderParameters( )`

KeyStore.CallbackHandlerProtection java.security

Java 5.0

This class is a `KeyStore.ProtectionParameter` implementation that wraps a `javax.security.auth.callback.CallbackHandler` for prompting the user for a password or other authentication credentials.

```
public static class KeyStore.CallbackHandlerProtection
        implements KeyStore.ProtectionParameter {
// Public Constructors
     public CallbackHandlerProtection(javax.security.auth.callback.
        CallbackHandler handler);
// Public Instance Methods
     public javax.security.auth.callback.CallbackHandler getCallbackHandler( );
}
```

KeyStore.Entry java.security

Java 5.0

This marker interface represents an entry in a `KeyStore`.

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

```
public interface KeyStore.Entry {
}
```

Implementations

`KeyStore.PrivateKeyEntry`, `KeyStore.SecretKeyEntry`, `KeyStore.TrustedCertificateEntry`

Passed To

`KeyStore.setEntry( )`, `KeyStoreSpi.engineSetEntry( )`

Returned By

`KeyStore.getEntry( )`, `KeyStoreSpi.engineGetEntry( )`

KeyStore.LoadStoreParameter java.security

Java 5.0

This interface represents an object passed to the `load( )` or `store( )` methods of `KeyStore`. An implementation must be able to return a `KeyStore.ProtectionParameter`.

```
public interface KeyStore.LoadStoreParameter {
// Public Instance Methods
     KeyStore.ProtectionParameter getProtectionParameter( );
}
```

Passed To

`KeyStore.{load( ), store( )}`, `KeyStoreSpi.{engineLoad( ), engineStore( )}`

KeyStore.PasswordProtection java.security

Java 5.0

This class is a `KeyStore.ProtectionParameter` implementation that wraps a password specified as a `char[ ]`. Note that `getPassword( )` returns a reference to the internal array, not a clone of it. The `destroy( )` method zeros out this array.

```
public static class KeyStore.PasswordProtection
        implements javax.security.auth.Destroyable, KeyStore.ProtectionParameter {
// Public Constructors
     public PasswordProtection(char[ ] password);
// Public Instance Methods
     public char[ ] getPassword( );           synchronized
// Methods Implementing Destroyable
     public void destroy( )
        throws javax.security.auth.DestroyFailedException;                    synchronized
```


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

```
    public boolean isDestroyed( );          synchronized
}
```

KeyStore.PrivateKeyEntry java.security

Java 5.0

This `KeyStore.Entry` implementation represents a private key. `getPrivateKey( )` returns the key. `getCertificateChain( )` returns the certificate chain of the corresponding public key. The first element of the returned array is the certificate of the ultimate certificate authority (CA). This "end entity" certificate is also available through the `getCertificate( )` method.

```
public static final class KeyStore.PrivateKeyEntry implements KeyStore.Entry {
// Public Constructors
    public PrivateKeyEntry(PrivateKey privateKey, java.security.cert.
       Certificate[ ] chain);
// Public Instance Methods
    public java.security.cert.Certificate getCertificate( );
    public java.security.cert.Certificate[ ] getCertificateChain( );
    public PrivateKey getPrivateKey( );
// Public Methods Overriding Object
    public String toString( );
}
```

KeyStore.ProtectionParameter java.security

Java 5.0

This marker interface should be implemented by classes that provide some form of protection for the entries in a `KeyStore`.

```
public interface KeyStore.ProtectionParameter {
}
```

Implementations

`KeyStore.CallbackHandlerProtection`, `KeyStore.PasswordProtection`

Passed To

`KeyStore.{getEntry( ),setEntry( )}`, `KeyStore.Builder.newInstance( )`, `KeyStoreSpi.{engineGetEntry( ), engineSetEntry( )}`

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

Returned By

`KeyStore.Builder.getProtectionParameter( )`, `KeyStore.LoadStoreParameter.getProtectionParameter( )`

KeyStore.SecretKeyEntry java.security

Java 5.0

This `KeyStore.Entry` implementation represents a secret key. `getSecretKey( )` returns the key as a `javax.crypto.SecretKey`.

```
public static final class KeyStore.SecretKeyEntry implements KeyStore.Entry {
// Public Constructors
     public SecretKeyEntry(javax.crypto.SecretKey secretKey);
// Public Instance Methods
     public javax.crypto.SecretKey getSecretKey( );
// Public Methods Overriding Object
     public String toString( );
}
```

KeyStore.TrustedCertificateEntry java.security

Java 5.0

This implementation of `KeyStore.Entry` represents a certificate that contains and certifies a public key. `getTrustedCertificate( )` returns the certificate.

```
public static final class KeyStore.TrustedCertificateEntry
        implements KeyStore.Entry {
// Public Constructors
     public TrustedCertificateEntry(java.security.cert.Certificate trustedCert);
// Public Instance Methods
     public java.security.cert.Certificate getTrustedCertificate( );
// Public Methods Overriding Object
     public String toString( );
}
```

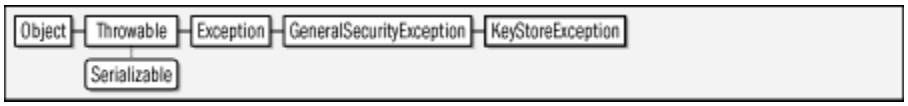
KeyStoreException java.security

Java 1.2 ***serializable checked***

Signals a problem with a `KeyStore`.

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

Figure 14-23. java.security.KeyStoreException




```
public class KeyStoreException extends GeneralSecurityException {
// Public Constructors
     public KeyStoreException( );
5.0  public KeyStoreException(Throwable cause);
     public KeyStoreException(String msg);
5.0  public KeyStoreException(String message, Throwable cause);
}
```

Thrown By

Too many methods to list.

KeyStoreSpi java.security

Java 1.2

This abstract class defines the service-provider interface for `KeyStore`. A security provider must implement a concrete subclass of this class for each `KeyStore` type it supports. Applications never need to use or subclass this class.

```
public abstract class KeyStoreSpi {
// Public Constructors
     public KeyStoreSpi( );
// Public Instance Methods
     public abstract java.util.Enumeration<String> engineAliases( );
     public abstract boolean engineContainsAlias(String alias);
     public abstract void engineDeleteEntry(String alias)
        throws KeyStoreException;
5.0  public boolean engineEntryInstanceOf(String alias, Class<?
        extends KeyStore.Entry> entryClass);
     public abstract java.security.cert.Certificate engineGetCertificate
        (String alias);
     public abstract String engineGetCertificateAlias(java.security.cert.
        Certificate cert);
     public abstract java.security.cert.Certificate[ ] engineGetCertificateChain
        (String alias);
     public abstract java.util.Date engineGetCreationDate(String alias);
5.0  public KeyStore.Entry engineGetEntry(String alias,
        KeyStore.ProtectionParameter protParam)
        throws KeyStoreException, NoSuchAlgorithmException, UnrecoverableEntryException;
     public abstract Key engineGetKey(String alias, char[ ] password)
        throws NoSuchAlgorithmException, UnrecoverableKeyException;
     public abstract boolean engineIsCertificateEntry(String alias);
     public abstract boolean engineIsKeyEntry(String alias);
5.0  public void engineLoad(KeyStore.LoadStoreParameter param)
        throws java.io.IOException, NoSuchAlgorithmException,
        java.security.cert.CertificateException;
     public abstract void engineLoad(java.io.InputStream stream, char[ ] password)
        throws java.io.IOException, NoSuchAlgorithmException,
        java.security.cert.CertificateException;
     public abstract void engineSetCertificateEntry(String alias,
        java.security.cert.Certificate cert)
        throws KeyStoreException;
5.0  public void engineSetEntry(String alias, KeyStore.Entry entry,
        KeyStore.ProtectionParameter protParam)
```


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

```
        throws KeyStoreException;
    public abstract void engineSetKeyEntry(String alias, byte[ ] key,
        java.security.cert.Certificate[ ] chain)
        throws KeyStoreException;
    public abstract void engineSetKeyEntry(String alias, Key key,
        char[ ] password, java.security.cert.Certificate[ ] chain)
        throws KeyStoreException;
    public abstract int engineSize( );
5.0 public void engineStore(KeyStore.LoadStoreParameter param)
        throws java.io.IOException, NoSuchAlgorithmException,
        java.security.cert.CertificateException;
    public abstract void engineStore(java.io.OutputStream stream,
        char[ ] password)
        throws java.io.IOException, NoSuchAlgorithmException,
        java.security.cert.CertificateException;
}
```

Passed To

`KeyStore.KeyStore( )`

MessageDigest java.security

Java 1.1

This class computes a message digest (also known as a cryptographic checksum) for an arbitrary sequence of bytes. Obtain a `MessageDigest` object by calling one of the static `getInstance( )` factory methods and specifying the desired algorithm (e.g., SHA or MD5) and, optionally, the desired provider. Next, specify the data to be digested by calling any of the `update( )` methods one or more times. Prior to Java 5.0, you must pass a `byte[ ]` to `update( )`. In Java 5.0 and later, however, you can also use a `java.nio.ByteBuffer`. This facilitates the computation of message digests when using the New I/O API.

After you pass data to `update( )`, call `digest( )`, which computes the message digest and returns it as an array of bytes. If you have only one array of bytes to be digested, you can pass it directly to `digest( )` and skip the `update( )` step. When you call `digest( )`, the `MessageDigest( )` object is reset and is then ready to compute a new digest. You can also explicitly reset a `MessageDigest` without computing the digest by calling `reset( )`. To compute a digest for part of a message without resetting the `MessageDigest`, clone the `MessageDigest` and call `digest( )` on the cloned copy. Note that not all implementations are cloneable, so the `clone( )` method may throw an exception.

The `MessageDigest` class is often used in conjunction with `DigestInputStream` and `DigestOutputStream`, which automate the `update( )` calls for you.

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

Figure 14-24. java.security.MessageDigest


```
public abstract class MessageDigest extends MessageDigestSpi {
// Protected Constructors
     protected MessageDigest(String algorithm);
// Public Class Methods
     public static MessageDigest getInstance(String algorithm)
        throws NoSuchAlgorithmException;
     public static MessageDigest getInstance(String algorithm,
        String provider)
        throws NoSuchAlgorithmException, NoSuchProviderException;
1.4  public static MessageDigest getInstance(String algorithm, Provider provider)
throws NoSuchAlgorithmException;
     public static boolean isEqual(byte[ ] digesta, byte[ ] digestb);
// Public Instance Methods
     public byte[ ] digest( );
     public byte[ ] digest(byte[ ] input);
1.2  public int digest(byte[ ] buf, int offset, int len)
        throws DigestException;
     public final String getAlgorithm( );
1.2  public final int getDigestLength( );
1.2  public final Provider getProvider( );
     public void reset( );
     public void update(byte input);
     public void update(byte[ ] input);
5.0  public final void update(java.nio.ByteBuffer input);
     public void update(byte[ ] input, int offset, int len);
// Public Methods Overriding MessageDigestSpi
     public Object clone( ) throws CloneNotSupportedException;
// Public Methods Overriding Object
     public String toString( );
}
```

Passed To

`DigestInputStream.{DigestInputStream( ),setMessageDigest( )}`,
`DigestOutputStream.{DigestOutputStream( ),setMessageDigest( )}`

Returned By

`DigestInputStream.getMessageDigest( )`,
`DigestOutputStream.getMessageDigest( )`

Type Of

`DigestInputStream.digest`, `DigestOutputStream.digest`

MessageDigestSpi java.security

Java 1.2

This abstract class defines the service-provider interface for `MessageDigest`. A security provider must implement a concrete subclass of this class for each message-digest algorithm it supports. Applications never need to use or subclass this class.

```
public abstract class MessageDigestSpi {
// Public Constructors
     public MessageDigestSpi( );
// Public Methods Overriding Object
```


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

```
    public Object clone( ) throws CloneNotSupportedException;
// Protected Instance Methods
    protected abstract byte[ ] engineDigest( );
    protected int engineDigest(byte[ ] buf, int offset, int len)
        throws DigestException;
    protected int engineGetDigestLength( );              constant
    protected abstract void engineReset( );
    protected abstract void engineUpdate(byte input);
5.0 protected void engineUpdate(java.nio.ByteBuffer input);
    protected abstract void engineUpdate(byte[ ] input, int offset, int len);
}
```

Subclasses

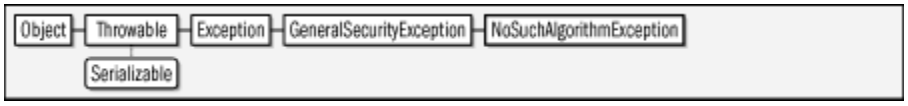
`MessageDigest`

NoSuchAlgorithmException java.security

Java 1.1 ***serializable checked***

Signals that a requested cryptographic algorithm is not available. Thrown by `getInstance( )` factory methods throughout the `java.security` package.

Figure 14-25. java.security.NoSuchAlgorithmException




```
public class NoSuchAlgorithmException extends GeneralSecurityException {
// Public Constructors
    public NoSuchAlgorithmException( );
5.0 public NoSuchAlgorithmException(Throwable cause);
    public NoSuchAlgorithmException(String msg);
5.0 public NoSuchAlgorithmException(String message, Throwable cause);
}
```

Thrown By

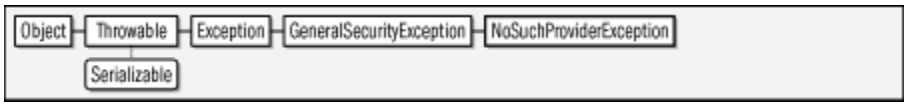
Too many methods to list.

NoSuchProviderException java.security

Java 1.1 ***serializable checked***

Signals that a requested cryptographic service provider is not available. Thrown by `getInstance( )` factory methods throughout the `java.security` package.

Figure 14-26. java.security.NoSuchProviderException




```
public class NoSuchProviderException extends GeneralSecurityException {
// Public Constructors
     public NoSuchProviderException( );
     public NoSuchProviderException(String msg);
}
```

Thrown By

Too many methods to list.

Permission	**java.security**
Java 1.2	***serializable permission***

This abstract class represents a system resource, such as a file in the filesystem, or a system capability, such as the ability to accept network connections. Concrete subclasses of `Permission`, such as `java.io.FilePermission` and `java.net.SocketPermission`, represent specific types of resources. `Permission` objects are used by system code that is requesting access to a resource. They are also used by `Policy` objects that grant access to resources. The `AccessController.checkPermission( )` method considers the source of the currently running Java code, determines the set of permissions that are granted to that code by the current `Policy`, and then checks to see whether a specified `Permission` object is included in that set. As of Java 1.2, this is the fundamental Java access-control mechanism.

Each permission has a name (sometimes called the *target*) and, optionally, a comma-separated list of actions. For example, the name of a `FilePermission` is the name of the file or directory for which permission is being granted. The actions associated with this permission might be "read"; "write"; or "read,write". The interpretation of the name and action strings is entirely up to the implementation of `Permission`. A number of implementations support the use of wildcards; for example, a `FilePermission` can have a name of "/tmp/*", which represents access to any files in a */tmp* directory. Permission objects must be immutable, so an implementation must never define a `setName( )` or `setActions( )` method.

One of the most important abstract methods defined by `Permission` is `implies( )`. This method must return `true` if this `Permission` implies another `Permission`. For example, if an application requests a `FilePermission` with name "/tmp/test" and action

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

"read", and the current security `Policy` grants a `FilePermission` with name "/tmp/*" and actions "read,write", the request is granted because the requested permission is implied by the granted one.

In general, only system-level code needs to work directly with `Permission` and its concrete subclasses. System administrators who are configuring security policies need to understand the various `Permission` subclasses. Applications that want to extend the Java access-control mechanism to provide customized access control to their own resources should subclass `Permission` to define custom permission types.

Figure 14-27. java.security.Permission


```
public abstract class Permission implements Guard, Serializable {
// Public Constructors
     public Permission(String name);
// Public Instance Methods
     public abstract String getActions( );
     public final String getName( );
     public abstract boolean implies(Permission permission);
     public PermissionCollection newPermissionCollection( );  constant
// Methods Implementing Guard
     public void checkGuard(Object object) throws SecurityException;
// Public Methods Overriding Object
     public abstract boolean equals(Object obj);
     public abstract int hashCode( );
     public String toString( );
}
```

Subclasses

`java.io.FilePermission`, `java.net.SocketPermission`, `AllPermission`, `BasicPermission`, `UnresolvedPermission`, `javax.security.auth.PrivateCredentialPermission`, `javax.security.auth.kerberos.ServicePermission`

Passed To

Too many methods to list.

Returned By

`java.net.HttpURLConnection.getPermission( )`, `java.net.URLConnection.getPermission( )`, `AccessControlException.getPermission( )`

PermissionCollection — java.security

Java 1.2 — ***serializable***

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

This class is used by `Permissions` to store a collection of `Permission` objects that are all the same type. Like the `Permission` class itself, `PermissionCollection` defines an `implies( )` method that can determine whether a requested `Permission` is implied by any of the `Permission` objects in the collection. Some `Permission` types may require a custom `PermissionCollection` type in order to correctly implement the `implies( )` method. In this case, the `Permission` subclass should override `newPermissionCollection( )` to return a `Permission` of the appropriate type. `PermissionCollection` is used by system code that manages security policies. Applications rarely need to use it.

Figure 14-28. java.security.PermissionCollection


```
public abstract class PermissionCollection implements Serializable {
// Public Constructors
     public PermissionCollection( );
// Public Instance Methods
     public abstract void add(Permission permission);
     public abstract java.util.Enumeration<Permission> elements( );
     public abstract boolean implies(Permission permission);
     public boolean isReadOnly( );
     public void setReadOnly( );
// Public Methods Overriding Object
     public String toString( );
}
```

Subclasses

`Permissions`

Passed To

`ProtectionDomain.ProtectionDomain( )`

Returned By

Too many methods to list.

Permissions — java.security

Java 1.2 — ***serializable***

This class stores an arbitrary collection of `Permission` objects. When `Permission` objects are added with the `add( )` method, they are grouped into an internal set of `PermissionCollection` objects that contain only a single type of `Permission`. Use the `elements( )` method to obtain an `Enumeration` of the `Permission` objects in the collection. Use `implies( )` to determine if a specified `Permission` is implied by any of the `Permission` objects in the collection. `Permissions` is used by system code that manages security policies. Applications rarely need to use it.

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

Figure 14-29. java.security.Permissions


```
public final class Permissions extends PermissionCollection
        implements Serializable {
// Public Constructors
     public Permissions( );
// Public Methods Overriding PermissionCollection
     public void add(Permission permission);
     public java.util.Enumeration<Permission> elements( );
     public boolean implies(Permission permission);
}
```

Policy java.security

Java 1.2

This class represents a security policy that determines the permissions granted to code based on its source and signers, and, in Java 1.4 and later, based on the user on whose behalf that code is running. There is only a single `Policy` in effect at any one time. Obtain the system policy by calling the static `getPolicy( )` method. Code that has appropriate permissions can specify a new system policy by calling `setPolicy( )`. The `refresh( )` method is a request to a `Policy` object to update its state (for example, by rereading its configuration file). The `Policy` class is used primarily by system-level code. Applications should not need to use this class unless they implement some kind of custom access-control mechanism.

Prior to Java 1.4, this class provides a mapping from `CodeSource` objects to `PermissionCollection` objects. `getPermissions( )` is the central `Policy` method; it evaluates the `Policy` for a given `CodeSource` and returns an appropriate `PermissionCollection` representing the static set of permissions available to code from that source.

As of Java 1.4, you can use a `ProtectionDomain` object to encapsulate a `CodeSource` and a set of users on whose behalf the code is running. In this release, there is a new `getPermissions( )` method that returns a `PermissionsCollection` appropriate for the specified `ProtectionDomain`. In addition, there is a new `implies( )` method that dynamically queries the `Policy` to see if the specified permission is granted to the specific `ProtectionDomain`.

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

```
public abstract class Policy {
// Public Constructors
     public Policy( );
// Public Class Methods
     public static java.security.Policy getPolicy( );
     public static void setPolicy(java.security.Policy p);
// Public Instance Methods
     public abstract PermissionCollection getPermissions(CodeSource codesource);
1.4  public PermissionCollection getPermissions(ProtectionDomain domain);
1.4  public boolean implies(ProtectionDomain domain, Permission permission);
     public abstract void refresh( );
}
```

Principal java.security

Java 1.1

This interface represents any entity that may serve as a principal in a cryptographic transaction of any kind. A `Principal` may represent an individual, a computer, or an organization, for example.

```
public interface Principal {
// Public Instance Methods
     boolean equals(Object another);
     String getName( );
     int hashCode( );
     String toString( );
}
```

Implementations

`Identity`, `javax.security.auth.kerberos.KerberosPrincipal`, `javax.security.auth.x500.X500Principal`

Passed To

`IdentityScope.getIdentity( )`, `ProtectionDomain.ProtectionDomain( )`, `javax.net.ssl.X509ExtendedKeyManager.{chooseEngineClientAlias( ), chooseEngineServerAlias( )}`, `javax.net.ssl.X509KeyManager.{chooseClientAlias( ), chooseServerAlias( ), getClientAliases( ), getServerAliases( )}`

Returned By

`java.net.SecureCacheResponse.{getLocalPrincipal( ), getPeerPrincipal( )}`, `java.security.Certificate.{getGuarantor( ), getPrincipal( )}`, `ProtectionDomain.getPrincipals( )`, `java.security.cert.X509Certificate.{getIssuerDN( ), getSubjectDN( )}`, `java.security.cert.X509CRL.getIssuerDN( )`, `javax.net.ssl.HandshakeCompletedEvent.{getLocalPrincipal( ), getPeerPrincipal( )}`, `javax.net.ssl.HttpsURLConnection.`

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

```
{getLocalPrincipal( ),getPeerPrincipal( )},
javax.net.ssl.SSLSession.{getLocalPrincipal( ),
getPeerPrincipal( )}
```

PrivateKey java.security

Java 1.1 ***serializable***

This interface represents a private cryptographic key. It extends the `Key` interface, but does not add any new methods. The interface exists in order to create a strong distinction between private and public keys. See also `PublicKey`.

Figure 14-30. java.security.PrivateKey


```
public interface PrivateKey extends Key {
// Public Constants
1.2  public static final long serialVersionUID;  =6034044314589513430
}
```

Implementations

```
java.security.interfaces.DSAPrivateKey,
java.security.interfaces.ECPrivateKey,
java.security.interfaces.RSAPrivateKey,
javax.crypto.interfaces.DHPrivateKey
```

Passed To

```
KeyPair.KeyPair( ),KeyStore.PrivateKeyEntry.PrivateKeyEntry( ),
Signature.initSign( ),SignatureSpi.engineInitSign( ),
SignedObject.SignedObject( ),
javax.security.auth.x500.X500PrivateCredential.X500PrivateCreden
tial( )
```

Returned By

```
KeyFactory.generatePrivate( ),
KeyFactorySpi.engineGeneratePrivate( ),KeyPair.getPrivate( ),
KeyStore.PrivateKeyEntry.getPrivateKey( ),Signer.getPrivateKey( ),
javax.net.ssl.X509KeyManager.getPrivateKey( ),
javax.security.auth.x500.X500PrivateCredential.getPrivateKey( )
```

PrivilegedAction<T> java.security

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

Java 1.2

This interface defines a block of code (the `run( )` method) that is to be executed as privileged code by the `AccessController.doPrivileged( )` method. In Java 5.0 this interface is generic and the type variable *`T`* represents the return type of the `run( )` method. When privileged code is run with the `doPrivileged( )` method, the `AccessController` looks only at the permissions of the immediate caller, not the permissions of the entire call stack. The immediate caller is typically fully trusted system code that has a full set of permissions, and therefore the privileged code runs with that full set of permissions, even if the system code is invoked by untrusted code with no permissions whatsoever.

Privileged code is typically required only when you are writing a trusted system library (such as a Java extension package) that must read local files or perform other restricted actions, even when called by untrusted code. For example, a class that must call `System.loadLibrary( )` to load native methods should make the call to `loadLibrary( )` within the `run( )` method of a `PrivilegedAction`. If your privileged code may throw a checked exception, implement it in the `run( )` method of a `PrivilegedExceptionAction` instead.

Be very careful when implementing this interface. To minimize the possibility of security holes, keep the body of the `run( )` method as short as possible.

```
public interface PrivilegedAction<T> {
// Public Instance Methods
     T run( );
}
```

Passed To

`AccessController.doPrivileged( )`,
`java.util.concurrent.Executors.callable( )`,
`javax.security.auth.Subject.{doAs( ),doAsPrivileged( )}`

PrivilegedActionException **java.security**

Java 1.2 ***serializable checked***

This exception class is a wrapper around an arbitrary `Exception` thrown by a `PrivilegedExceptionAction` executed by the `AccessController.doPrivileged( )` method. Use `getException( )` to obtain

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

the wrapped `Exception` object. Or, in Java 1.4 and later, use the more general `getCause( )` method.

Figure 14-31. java.security.PrivilegedActionException


```
public class PrivilegedActionException extends Exception {
// Public Constructors
     public PrivilegedActionException(Exception exception);
// Public Instance Methods
     public Exception getException( );
// Public Methods Overriding Throwable
1.4  public Throwable getCause( );
1.3  public String toString( );
}
```

Thrown By

`AccessController.doPrivileged( ),javax.security.auth.Subject.{doAs( ),doAsPrivileged( )}`

PrivilegedExceptionAction<T> java.security

Java 1.2

This interface is like `PrivilegedAction`, except that its `run( )` method may throw an exception. See `PrivilegedAction` for details.

```
public interface PrivilegedExceptionAction<T> {
// Public Instance Methods
     T run( ) throws Exception;
}
```

Passed To

`AccessController.doPrivileged( ),`
`java.util.concurrent.Executors.callable( ),`
`javax.security.auth.Subject.{doAs( ),doAsPrivileged( )}`

ProtectionDomain java.security

Java 1.2

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

This class represents a "protection domain": the set of permissions associated with code based on its source, and optionally, the identities of the users on whose behalf the code is running. Use the `getProtectionDomain( )` of a `Class` object to obtain the `ProtectionDomain` that the class is part of.

Prior to Java 1.4, a `ProtectionDomain` simply associates a `CodeSource` with the `PermissionCollection` granted to code from that source by a `Policy`. The set of permissions is static, and the `implies( )` method checks to see whether the specified `Permission` is implied by any of the permissions granted to this `ProtectionDomain`.

In Java 1.4 and later, a `ProtectionDomain` can also be created with the four-argument constructor which associates a `PermissionCollection` with a `ClassLoader` and an array of `Principal` objects in addition to a `CodeSource`. A `ProtectionDomain` of this sort represents permisssions granted to code loaded from a specified source, through a specified class loader, and running under the auspices of one or more specified principals. When a `ProtectionDomain` is instantiated with this four-argument constructor, the `PermissionCollection` is not static, and the `implies( )` method calls the `implies( )` method of the current `Policy` object before checking the specified collection of permissions. This allows security policies to be updated (for example to add new permissions for specific users) without having to restart long-running programs such as servers.

```
public class ProtectionDomain {
// Public Constructors
     public ProtectionDomain(CodeSource codesource,
        PermissionCollection permissions);
1.4  public ProtectionDomain(CodeSource codesource,
        PermissionCollection permissions, ClassLoader classloader,
Principal[ ] principals);
// Public Instance Methods
1.4  public final ClassLoader getClassLoader( );
     public final CodeSource getCodeSource( );
     public final PermissionCollection getPermissions( );
1.4  public final Principal[ ] getPrincipals( );
     public boolean implies(Permission permission);
// Public Methods Overriding Object
     public String toString( );
}
```

Passed To

`ClassLoader.defineClass( )`, `java.lang.instrument.ClassFileTransformer.transform( )`, `AccessControlContext.AccessControlContext( )`, `DomainCombiner.combine( )`, `java.security.Policy.{getPermissions( ), implies( )}`, `javax.security.auth.SubjectDomainCombiner.combine( )`

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

Returned By

`Class.getProtectionDomain( )`, `DomainCombiner.combine( )`, `javax.security.auth.SubjectDomainCombiner.combine( )`

Provider — java.security

Java 1.1 — ***cloneable serializable collection***

This class represents a security provider. It specifies class names for implementations of one or more algorithms for message digests, digital signatures, key generation, key conversion, key management, secure random number generation, certificate conversion, and algorithm parameter management. The `getName( )`, `getVersion( )`, and `getInfo( )` methods return information about the provider. `Provider` inherits from `Properties` and maintains a mapping of property names to property values. These name/value pairs specify the capabilities of the `Provider` implementation. Each property name has the form:

```
service_type.algorithm_name
```

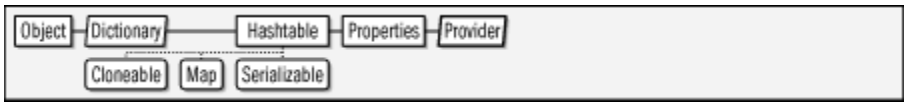
The corresponding property value is the name of the class that implements the named algorithm. For example, say a `Provider` defines properties named "Signature.DSA", "MessageDigest.MD5", and "KeyStore.JKS". The values of these properties are the class names of `SignatureSpi`, `MessageDigestSpi`, and `KeyStoreSpi` implementations. Other properties defined by a `Provider` are used to provide aliases for algorithm names. For example, the property `Alg.Alias.MessageDigest.SHA1` might have the value "SHA", meaning that the algorithm name "SHA1" is an alias for "SHA".

In Java 5.0, the individual services provided by a `Provider` are described by the nested `Service` class, and various methods for querying and setting the `Service` objects of a `Provider` are available.

Security providers are installed in an implementation-dependent way. For Sun's implementation, the *${java.home}/lib/security/java.security* file specifies the class names of all installed `Provider` implementations. An application can also install its own custom `Provider` with the `addProvider( )` and `insertProviderAt( )` methods of the `Security` class. Most applications do not need to use the `Provider` class directly. Typically, only security-provider implementors need to use the `Provider` class. Some applications may explicitly specify the name of a desired `Provider` when calling a static `getInstance( )` factory method, however. Only applications with the most demanding cryptographic needs require custom providers.

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

Figure 14-32. java.security.Provider




```
public abstract class Provider extends java.util.Properties {
// Protected Constructors
     protected Provider(String name, double version, String info);
// Nested Types
5.0  public static class Service;
// Public Instance Methods
     public String getInfo( );
     public String getName( );
5.0  public Provider.Service getService(String type,
        String algorithm);     synchronized
5.0  public java.util.Set<Provider.Service> getServices( );     synchronized
     public double getVersion( );
// Public Methods Overriding Properties
1.2  public void load(java.io.InputStream inStream) throws java.io.IOException;     synchronized
// Public Methods Overriding Hashtable
1.2  public void clear( );     synchronized
1.2  public java.util.Set<java.util.Map.Entry<Object,
        Object>> entrySet( );     synchronized
1.2  public java.util.Set<Object> keySet( );
1.2  public Object put(Object key, Object value);       synchronized
1.2  public void putAll(java.util.Map<?,?> t);             synchronized
1.2  public Object remove(Object key);              synchronized
     public String toString( );
1.2  public java.util.Collection<Object> values( );
// Protected Instance Methods
5.0  protected void putService(Provider.Service s);        synchronized
5.0  protected void removeService(Provider.Service s);      synchronized
}
```

Subclasses

`AuthProvider`

Passed To

Too many methods to list.

Returned By

Too many methods to list.

Provider.Service java.security

Java 5.0

This nested class represents a single service (such as a hash algorithm) provided by a security `Provider`. The various methods return information about the service, including the name of the implementing class.

```
public static class Provider.Service {
// Public Constructors
     public Service(Provider provider, String type, String algorithm,
        String className, java.util.List<String> aliases,
        java.util.Map<String,String> attributes);
// Public Instance Methods
     public final String getAlgorithm( );
     public final String getAttribute(String name);
```


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

```
    public final String getClassName( );
    public final Provider getProvider( );
    public final String getType( );
    public Object newInstance(Object constructorParameter)
       throws NoSuchAlgorithmException;
    public boolean supportsParameter(Object parameter);
// Public Methods Overriding Object
    public String toString( );
}
```

Passed To

`Provider.{putService( ),removeService( )}`

Returned By

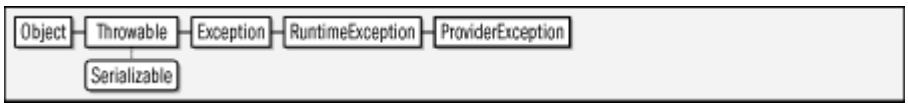
`Provider.getService( )`

ProviderException java.security

Java 1.1 ***serializable unchecked***

Signals that an exception has occurred inside a cryptographic service provider. Note that `ProviderException` extends `RuntimeException` and is therefore an unchecked exception that may be thrown from any method without being declared.

Figure 14-33. java.security.ProviderException




```
public class ProviderException extends RuntimeException {
// Public Constructors
    public ProviderException( );
5.0 public ProviderException(Throwable cause);
    public ProviderException(String s);
5.0 public ProviderException(String message, Throwable cause);
}
```

PublicKey java.security

Java 1.1 ***serializable***

This interface represents a public cryptographic key. It extends the `Key` interface, but does not add any new methods. The interface exists in order to create a strong distinction between public and private keys. See also `PrivateKey`.

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

Figure 14-34. java.security.PublicKey


```
public interface PublicKey extends Key {
// Public Constants
1.2  public static final long serialVersionUID;  =7187392471159151072
}
```

Implementations

`java.security.interfaces.DSAPublicKey`, `java.security.interfaces.ECPublicKey`, `java.security.interfaces.RSAPublicKey`, `javax.crypto.interfaces.DHPublicKey`

Passed To

`Identity.setPublicKey( )`, `IdentityScope.getIdentity( )`, `KeyPair.KeyPair( )`, `Signature.initVerify( )`, `SignatureSpi.engineInitVerify( )`, `SignedObject.verify( )`, `java.security.cert.Certificate.verify( )`, `java.security.cert.PKIXCertPathBuilderResult.PKIXCertPathBuilderResult( )`, `java.security.cert.PKIXCertPathValidatorResult.PKIXCertPathValidatorResult( )`, `java.security.cert.TrustAnchor.TrustAnchor( )`, `java.security.cert.X509CertSelector.setSubjectPublicKey( )`, `java.security.cert.X509CRL.verify( )`

Returned By

`java.security.Certificate.getPublicKey( )`, `Identity.getPublicKey( )`, `KeyFactory.generatePublic( )`, `KeyFactorySpi.engineGeneratePublic( )`, `KeyPair.getPublic( )`, `java.security.cert.Certificate.getPublicKey( )`, `java.security.cert.PKIXCertPathValidatorResult.getPublicKey( )`, `java.security.cert.TrustAnchor.getCAPublicKey( )`, `java.security.cert.X509CertSelector.getSubjectPublicKey( )`

SecureClassLoader — java.security

Java 1.2

This class adds protected methods to those defined by `ClassLoader`. The `defineClass( )` method is passed the bytes of a class file as a `byte[ ]` or, in Java 5.0, as a `ByteBuffer` and a `CodeSource` object that represents the source of that class. It calls the `getPermissions( )` method to obtain a `PermissionCollection` for that

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

`CodeSource` and then uses the `CodeSource` and `PermissionCollection` to create a `ProtectionDomain`, which is passed to the `defineClass( )` method of its superclass.

The default implementation of the `getPermissions( )` method uses the default `Policy` to determine the appropriate set of permissions for a given code source. The value of `SecureClassLoader` is that subclasses can use its `defineClass( )` method to load classes without having to work explicitly with the `ProtectionDomain` and `Policy` classes. A subclass of `SecureClassLoader` can define its own security policy by overriding `getPermissions( )`. In Java 1.2 and later, any application that implements a custom class loader should do so by extending `SecureClassLoader`, instead of subclassing `ClassLoader` directly. Most applications can use `java.net.URLClassLoader`, however, and never have to subclass this class.

Figure 14-35. java.security.SecureClassLoader


```
public class SecureClassLoader extends ClassLoader {
// Protected Constructors
     protected SecureClassLoader( );
     protected SecureClassLoader(ClassLoader parent);
// Protected Instance Methods
5.0  protected final Class<?> defineClass(String name,
        java.nio.ByteBuffer b, CodeSource cs);
     protected final Class<?> defineClass(String name, byte[ ] b, int off,
        int len, CodeSource cs);
     protected PermissionCollection getPermissions(CodeSource codesource);
}
```

Subclasses

`java.net.URLClassLoader`

SecureRandom — java.security

Java 1.1 — ***serializable***

This class generates cryptographic-quality pseudorandom bytes. Although `SecureRandom` defines public constructors, the preferred technique for obtaining a `SecureRandom` object is to call one of the static `getInstance( )` factory methods, specifying the desired pseudorandom number-generation algorithm, and, optionally, the desired provider of that algorithm. Sun's implementation of Java ships with an algorithm named "SHA1PRNG" in the "SUN" provider.

Once you have obtained a `SecureRandom` object, call `nextBytes( )` to fill an array with pseudorandom bytes. You can also call any of the methods defined by the `Random`

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

superclass to obtain random numbers. The first time one of these methods is called, the `SecureRandom( )` method uses its `generateSeed( )` method to seed itself. If you have a source of random or very high-quality pseudorandom bytes, you may provide your own seed by calling `setSeed( )`. Repeated calls to `setSeed( )` augment the existing seed instead of replacing it. You can also call `generateSeed( )` to generate seeds for use with other pseudorandom generators. `generateSeed( )` may use a different algorithm than `nextBytes( )` and may produce higher-quality randomness, usually at the expense of increased computation time.

Figure 14-36. java.security.SecureRandom


```
public class SecureRandom extends java.util.Random {
// Public Constructors
     public SecureRandom( );
     public SecureRandom(byte[ ] seed);
// Protected Constructors
1.2  protected SecureRandom(SecureRandomSpi secureRandomSpi, Provider provider);
// Public Class Methods
1.2  public static SecureRandom getInstance(String algorithm)
        throws NoSuchAlgorithmException;
1.2  public static SecureRandom getInstance(String algorithm, String provider)
        throws NoSuchAlgorithmException, NoSuchProviderException;
1.4  public static SecureRandom getInstance(String algorithm, Provider provider)
        throws NoSuchAlgorithmException;
     public static byte[ ] getSeed(int numBytes);
// Public Instance Methods
1.2  public byte[ ] generateSeed(int numBytes);
5.0  public String getAlgorithm( );   default:"NativePRNG"
1.2  public final Provider getProvider( );
     public void setSeed(byte[ ] seed);       synchronized
// Public Methods Overriding Random
     public void nextBytes(byte[ ] bytes);    synchronized
     public void setSeed(long seed);
// Protected Methods Overriding Random
     protected final int next(int numBits);
}
```

Passed To

Too many methods to list.

Type Of

`SignatureSpi.appRandom`

SecureRandomSpi java.security

Java 1.2 ***serializable***

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

This abstract class defines the service-provider interface for `SecureRandom`. A security provider must implement a concrete subclass of this class for each pseudorandom number-generation algorithm it supports. Applications never need to use or subclass this class.

Figure 14-37. java.security.SecureRandomSpi


```
public abstract class SecureRandomSpi implements Serializable {
// Public Constructors
     public SecureRandomSpi( );
// Protected Instance Methods
     protected abstract byte[ ] engineGenerateSeed(int numBytes);
     protected abstract void engineNextBytes(byte[ ] bytes);
     protected abstract void engineSetSeed(byte[ ] seed);
}
```

Passed To

`SecureRandom.SecureRandom( )`

Security java.security

Java 1.1

This class defines static methods both for managing the list of installed security providers and for reading and setting the values of various properties used by the Java security system. It is essentially an interface to the *${java.home}/lib/security/java.security* properties file that is included in Sun's implementation of Java. Use `getProperty( )` and `setProperty( )` to query or set the value of security properties whose default values are stored in that file.

One of the important features of the *java.security* properties file is that it specifies a set of security provider implementations and a preference order in which they are to be used. `getProviders( )` returns an array of `Provider` objects, in the order they are specified in the file. In Java 1.3 and later, versions of this method exist that only return providers that implement the algorithm or algorithms specified in a `String` or `Map` object. You can also look up a single named `Provider` object by name with `getProvider( )`. Note that a provider name is the string returned by `getName( )` method of the `Provider` class, not the classname of the `Provider`.

You can alter the set of providers installed by default from the `java.security` file. Use `addProvider( )` to add a new `Provider` object to the list, placing it at the end of the list, with a lower preference than all other providers. Use `insertProviderAt( )` to insert a provider into the list at a specified position. Note that provider preference positions

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

are 1-based. Specify a position of 1 to make the provider the most preferred one. Finally, use `removeProvider( )` to remove a named provider.

In Java 1.4 and later, the `getAlgorithms` method returns a `Set` that includes the names of all supported algorithms (from any installed provider) for the specified "service". A service name specifies the category of security service you are querying. It is a case-insensitive value that has the same name as one of the key service classes from this package or security-related packages—for example, "Signature", "MessageDigest", and "KeyStore" (from this package) or "Cipher" (from the `javax.crypto` package).

```
public final class Security {
// No Constructor
// Public Class Methods
     public static int addProvider(Provider provider);
1.4  public static java.util.Set<String> getAlgorithms(String serviceName);
     public static String getProperty(String key);
     public static Provider getProvider(String name);
     public static Provider[ ] getProviders( );
1.3  public static Provider[ ] getProviders(java.util.Map<String,String> filter);
1.3  public static Provider[ ] getProviders(String filter);
     public static int insertProviderAt(Provider provider,
        int position);     synchronized
     public static void removeProvider(String name);       synchronized
     public static void setProperty(String key, String datum);
// Deprecated Public Methods
#    public static String getAlgorithmProperty(String algName, String propName);
}
```

SecurityPermission — java.security

Java 1.2 — ***serializable permission***

This class is a `Permission` subclass that represents access to various methods of the `Policy`, `Security`, `Provider`, `Signer`, and `Identity` objects. `SecurityPermission` objects are defined by a name only; they do not use a list of actions. Important `SecurityPermission` names are "getPolicy" and "setPolicy", which represent the ability query and set the system security policy by invoking the `Policy.getPolicy( )` and `Policy.setPolicy( )` methods. Applications do not typically need to use this class.

Figure 14-38. java.security.SecurityPermission


```
public final class SecurityPermission extends BasicPermission {
// Public Constructors
     public SecurityPermission(String name);
```


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

```
     public SecurityPermission(String name, String actions);
}
```

Signature — java.security

Java 1.1

This class computes or verifies a digital signature. Obtain a `Signature` object by calling one of the static `getInstance( )` factory methods and specifying the desired digital signature algorithm and, optionally, the desired provider of that algorithm. A *digital signature* is essentially a message digest encrypted by a public-key encryption algorithm. Thus, to specify a digital signature algorithm, you must specify both the digest algorithm and the encryption algorithm. The only algorithm supported by the default "SUN" provider is "SHA1withDSA".

Once you have obtained a `Signature` object, you must initialize it before you can create or verify a digital signature. To initialize a digital signature for creation, call `initSign( )` and specify the private key to be used to create the signature. To initialize a signature for verification, call `initVerify( )` and specify the public key of the signer. Once the `Signature` object has been initialized, call `update( )` one or more times to specify the data to be signed or verified. Prior to Java 5.0, the data must be specified as an array of bytes. In Java 5.0 and later, you can also pass a `ByteBuffer` to `update( )`, and this facilitates the use of the `Signature` class with the `java.nio` package.

Finally, to create a digital signature, call `sign( )`, passing a byte array into which the signature is stored. Or, pass the bytes of the digital signature to `verify( )`, which returns `true` if the signature is valid or `false` otherwise. After calling either `sign( )` or `verify( )`, the `Signature` object is reset internally and can be used to create or verify another signature.

Figure 14-39. java.security.Signature


```
public abstract class Signature extends SignatureSpi {
// Protected Constructors
     protected Signature(String algorithm);
// Protected Constants
     protected static final int SIGN;      =2
     protected static final int UNINITIALIZED;                          =0
     protected static final int VERIFY;    =3
// Public Class Methods
     public static Signature getInstance(String algorithm)
        throws NoSuchAlgorithmException;
```


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

```
1.4  public static Signature getInstance(String algorithm, Provider provider)
        throws NoSuchAlgorithmException;
     public static Signature getInstance(String algorithm, String provider)
throws NoSuchAlgorithmException, NoSuchProviderException;
// Public Instance Methods
     public final String getAlgorithm( );
1.4  public final AlgorithmParameters getParameters( );
1.2  public final Provider getProvider( );
     public final void initSign(PrivateKey privateKey)
        throws InvalidKeyException;
1.2  public final void initSign(PrivateKey privateKey, SecureRandom random)
        throws InvalidKeyException;
1.3  public final void initVerify(java.security.cert.Certificate certificate)
        throws InvalidKeyException;
     public final void initVerify(PublicKey publicKey)
        throws InvalidKeyException;
1.2  public final void setParameter(java.security.spec.
        AlgorithmParameterSpec params)
        throws InvalidAlgorithmParameterException;
     public final byte[ ] sign( ) throws SignatureException;
1.2  public final int sign(byte[ ] outbuf, int offset, int len) throws SignatureException;
5.0  public final void update(java.nio.ByteBuffer data) throws SignatureException;
     public final void update(byte b) throws SignatureException;
     public final void update(byte[ ] data) throws SignatureException;
     public final void update(byte[ ] data, int off, int len)
        throws SignatureException;
     public final boolean verify(byte[ ] signature) throws SignatureException;
1.4  public final boolean verify(byte[ ] signature, int offset, int length)
        throws SignatureException;
// Public Methods Overriding SignatureSpi
     public Object clone( ) throws CloneNotSupportedException;
// Public Methods Overriding Object
     public String toString( );
// Protected Instance Fields
     protected int state;
// Deprecated Public Methods
#    public final Object getParameter(String param)
        throws InvalidParameterException;
#    public final void setParameter(String param, Object value)
        throws InvalidParameterException;
}
```

Passed To

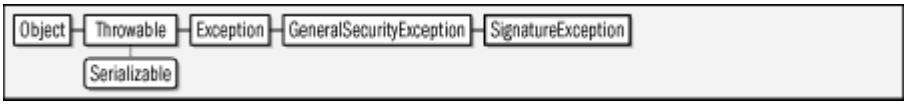
`SignedObject.{SignedObject( ),verify( )}`

SignatureException — java.security

Java 1.1 — ***serializable checked***

Signals a problem while creating or verifying a digital signature.

Figure 14-40. java.security.SignatureException




```
public class SignatureException extends GeneralSecurityException {
// Public Constructors
     public SignatureException( );
```


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

```
5.0  public SignatureException(Throwable cause);
     public SignatureException(String msg);
5.0  public SignatureException(String message, Throwable cause);
}
```

Thrown By

Too many methods to list.

SignatureSpi java.security

Java 1.2

This abstract class defines the service-provider interface for `Signature`. A security provider must implement a concrete subclass of this class for each digital signature algorithm it supports. Applications never need to use or subclass this class.

```
public abstract class SignatureSpi {
// Public Constructors
     public SignatureSpi( );
// Public Methods Overriding Object
     public Object clone( ) throws CloneNotSupportedException;
// Protected Instance Methods
1.4  protected AlgorithmParameters engineGetParameters( );
     protected abstract void engineInitSign(PrivateKey privateKey)
        throws InvalidKeyException;
     protected void engineInitSign(PrivateKey privateKey, SecureRandom random)
        throws InvalidKeyException;
     protected abstract void engineInitVerify(PublicKey publicKey)
        throws InvalidKeyException;
     protected void engineSetParameter(java.security.spec.
        AlgorithmParameterSpec params)
        throws InvalidAlgorithmParameterException;
     protected abstract byte[ ] engineSign( ) throws SignatureException;
     protected int engineSign(byte[ ] outbuf, int offset, int len)
        throws SignatureException;
5.0  protected void engineUpdate(java.nio.ByteBuffer input);
     protected abstract void engineUpdate(byte b) throws SignatureException;
     protected abstract void engineUpdate(byte[ ] b, int off, int len)
        throws SignatureException;
     protected abstract boolean engineVerify(byte[ ] sigBytes)
        throws SignatureException;
1.4  protected boolean engineVerify(byte[ ] sigBytes, int offset, int length)
        throws SignatureException;
// Protected Instance Fields
     protected SecureRandom appRandom;
// Deprecated Protected Methods
#    protected abstract Object engineGetParameter(String param)
        throws InvalidParameterException;
#    protected abstract void engineSetParameter(String param, Object value)
        throws InvalidParameterException;
}
```

Subclasses

`Signature`

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

SignedObject java.security

Java 1.2 ***serializable***

This class applies a digital signature to any serializable Java object. Create a `SignedObject` by specifying the object to be signed, the `PrivateKey` to use for the signature, and the `Signature` object to create the signature. The `SignedObject( )` constructor serializes the specified object into an array of bytes and creates a digital signature for those bytes.

After creation, a `SignedObject` is itself typically serialized for storage or transmission to another Java thread or process. Once the `SignedObject` is reconstituted, the integrity of the object it contains can be verified by calling `verify( )` and supplying the `PublicKey` of the signer and a `Signature` that performs the verification. Whether or not verification is performed or is successful, `getObject( )` can be called to deserialize and return the wrapped object.

Figure 14-41. java.security.SignedObject


```
public final class SignedObject implements Serializable {
// Public Constructors
     public SignedObject(Serializable object, PrivateKey signingKey,
        Signature signingEngine)
        throws java.io.IOException, InvalidKeyException, SignatureException;
// Public Instance Methods
     public String getAlgorithm( );
     public Object getObject( ) throws java.io.IOException,
        ClassNotFoundException;
     public byte[ ] getSignature( );
     public boolean verify(PublicKey verificationKey,
        Signature verificationEngine)
        throws InvalidKeyException, SignatureException;
}
```

Signer java.security

Java 1.1; Deprecated in 1.2 ***@Deprecated serializable***

This deprecated class was used in Java 1.1 to represent an entity or `Principal` that has an associated `PrivateKey` that enables it to create digital signatures. As of Java 1.2, this

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

class and the related `Identity` and `IdentityScope` classes have been replaced by `KeyStore` and `java.security.cert.Certificate`. See also `Identity`.

Figure 14-42. java.security.Signer


```
public abstract class Signer extends Identity {
// Public Constructors
     public Signer(String name);
     public Signer(String name, IdentityScope scope)
        throws KeyManagementException;
// Protected Constructors
     protected Signer( );
// Public Instance Methods
     public PrivateKey getPrivateKey( );
     public final void setKeyPair(KeyPair pair)
        throws InvalidParameterException, KeyException;
// Public Methods Overriding Identity
     public String toString( );
}
```

Timestamp java.security

Java 5.0 ***serializable***

An instance of this class is an immutable signed timestamp. `getTimestamp( )` returns the timestamp as a `java.util.Date`. `getSignerCertPath( )` returns the certificate path of the Timestamping Authority (TSA) that signed the object. `Timestamp` objects are used by the `CodeSigner` class.

Figure 14-43. java.security.Timestamp


```
public final class Timestamp implements Serializable {
// Public Constructors
     public Timestamp(java.util.Date timestamp,
        java.security.cert.CertPath signerCertPath);
// Public Instance Methods
     public java.security.cert.CertPath getSignerCertPath( );
     public java.util.Date getTimestamp( );
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
     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

`CodeSigner.CodeSigner( )`

Returned By

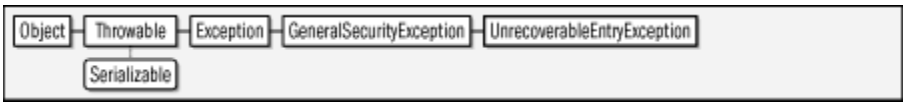
`CodeSigner.getTimestamp( )`

UnrecoverableEntryException java.security

Java 5.0 ***serializable checked***

An exception of this type is thrown if a `KeyStore.Entry` cannot be recovered from a `KeyStore`.

Figure 14-44. java.security.UnrecoverableEntryException




```
public class UnrecoverableEntryException extends GeneralSecurityException {
// Public Constructors
     public UnrecoverableEntryException( );
     public UnrecoverableEntryException(String msg);
}
```

Thrown By

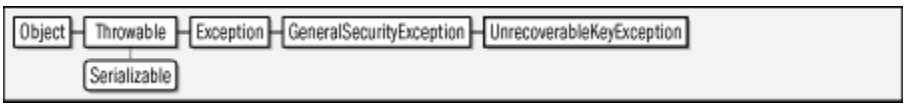
`KeyStore.getEntry( ),KeyStoreSpi.engineGetEntry( )`

UnrecoverableKeyException java.security

Java 1.2 ***serializable checked***

This exception is thrown if a `Key` cannot be retrieved from a `KeyStore`. This commonly occurs when an incorrect password is used.

Figure 14-45. java.security.UnrecoverableKeyException




```
public class UnrecoverableKeyException extends GeneralSecurityException {
// Public Constructors
     public UnrecoverableKeyException( );
     public UnrecoverableKeyException(String msg);
}
```


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

Thrown By

`KeyStore.getKey( ),KeyStoreSpi.engineGetKey( ), javax.net.ssl.KeyManagerFactory.init( ), javax.net.ssl.KeyManagerFactorySpi.engineInit( )`

UnresolvedPermission java.security

Java 1.2 ***serializable permission***

This class is used internally to provide a mechanism for delayed resolution of permissions (such as those whose implementation is in an external JAR file that has not been loaded yet). An `UnresolvedPermission` holds a representation of a `Permission` object that can later be used to create the actual `Permission` object. Java 5.0 adds methods to obtain details about the unresolved permission. Applications never need to use this class.

Figure 14-46. java.security.UnresolvedPermission


```
public final class UnresolvedPermission extends Permission
        implements Serializable {
// Public Constructors
     public UnresolvedPermission(String type, String name, String actions,
        java.security.cert.Certificate[ ] certs);
// Public Instance Methods
5.0  public String getUnresolvedActions( );
5.0  public java.security.cert.Certificate[ ] getUnresolvedCerts( );
5.0  public String getUnresolvedName( );
5.0  public String getUnresolvedType( );
// Public Methods Overriding Permission
     public boolean equals(Object obj);
     public String getActions( );
     public int hashCode( );
     public boolean implies(Permission p);                      constant
     public PermissionCollection newPermissionCollection( );
     public String toString( );
}
```

Package java.security.cert

Java 1.2

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

The `java.security.cert` package contains classes for working with identity certificates, certificate chains (also known as certification paths) and certificate revocation lists (CRLs). It defines generic `Certificate` and `CRL` classes and also `X509Certificate` and `X509CRL` classes that provide full support for standard X.509 certificates and CRLs. The `CertPath` class represents a certificate chain, and `CertPathValidator` provides the ability to validate a certificate chain. The `CertificateFactory` class serves as a certificate parser, providing the ability to convert a stream of bytes (or the base64 encoding of those bytes) into a `Certificate`, a `CertPath` or a `CRL` object. In addition to the algorithm-independent API of `CertificateFactory`, this package also defines low-level algorithm-specific classes for working with certificate chains using the PKIX standards.

This package replaces the deprecated `java.security.Certificate` interface, and it also replaces the deprecated `javax.security.cert` package used by early versions of the JAAS API before `javax.security.auth` and its subpackages were added to the core Java platform.

Interfaces

```
public interface CertPathBuilderResult extends Cloneable;
public interface CertPathParameters extends Cloneable;
public interface CertPathValidatorResult extends Cloneable;
public interface CertSelector extends Cloneable;
public interface CertStoreParameters extends Cloneable;
public interface CRLSelector extends Cloneable;
public interface PolicyNode;
public interface X509Extension;
```

Classes

```
public abstract class Certificate implements Serializable;
   public abstract class X509Certificate extends Certificate
   implements X509Extension;
public class CertificateFactory;
public abstract class CertificateFactorySpi;
public abstract class CertPath implements Serializable;
public class CertPathBuilder;
public abstract class CertPathBuilderSpi;
public class CertPathValidator;
public abstract class CertPathValidatorSpi;
public class CertStore;
public abstract class CertStoreSpi;
public class CollectionCertStoreParameters implements CertStoreParameters;
public abstract class CRL;
   public abstract class X509CRL extends CRL implements X509Extension;
public class LDAPCertStoreParameters implements CertStoreParameters;
public abstract class PKIXCertPathChecker implements Cloneable;
public class PKIXCertPathValidatorResult implements CertPathValidatorResult;
   public class PKIXCertPathBuilderResult extends PKIXCertPathValidatorResult
   implements CertPathBuilderResult;
public class PKIXParameters implements CertPathParameters;
   public class PKIXBuilderParameters extends PKIXParameters;
public class PolicyQualifierInfo;
public class TrustAnchor;
public class X509CertSelector implements CertSelector;
public abstract class X509CRLEntry implements X509Extension;
public class X509CRLSelector implements CRLSelector;
```


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

Protected Nested Types

```
protected static class Certificate.CertificateRep implements Serializable;
protected static class CertPath.CertPathRep implements Serializable;
```

Exceptions

```
public class CertificateException extends java.security.GeneralSecurityException;
   public class CertificateEncodingException extends CertificateException;
   public class CertificateExpiredException extends CertificateException;
   public class CertificateNotYetValidException extends CertificateException;
   public class CertificateParsingException extends CertificateException;
public class CertPathBuilderException
extends java.security.GeneralSecurityException;
public class CertPathValidatorException
extends java.security.GeneralSecurityException;
public class CertStoreException extends java.security.GeneralSecurityException;
public class CRLException extends java.security.GeneralSecurityException;
```

Certificate — java.security.cert

Java 1.2 — ***serializable***

This abstract class represents an public-key (or identity) certificate. A *certificate* is an object that contains the name of an entity and a public key for that entity. Certificates are issued by, and bear the digital signature of, a (presumably trusted) third party, typically a *certificate authority* (CA). By issuing and signing the certificate, the CA is certifying that, based on their research, the entity named on the certificate really is who they say they are and that the public key in the certificate really does belong to that entity. Sometimes the signer of a certificate is not a trusted CA, and the certificate is accompanied by the signer's certificate which may be signed by a CA, or by another untrusted intermediary who provides his or her own certificate. A "chain" of such certificates is known as a "certification path". See `CertPath` for further details.

Use a `CertificateFactory` to parse a stream of bytes into a `Certificate` object; `getEncoded( )` reverses this process. Use `verify( )` to verify the digital signature of the entity that issued the certificate. If the signature cannot be verified, the certificate should not be trusted. Call `getPublicKey( )` to obtain the `java.security.PublicKey` of the subject of the certificate. Note that this class does not define a method for obtaining the `Principal` that is associated with the `PublicKey`. That functionality is dependent on the type of the certificate. See `X509Certificate.getSubjectDN( )`, for example.

Do not confuse this class with the `java.security.Certificate` interface that was defined in Java 1.1 and has been deprecated in Java 1.2.

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

Figure 14-47. java.security.cert.Certificate


```
public abstract class Certificate implements Serializable {
// Protected Constructors
     protected Certificate(String type);
// Nested Types
1.3  protected static class CertificateRep implements Serializable;
// Public Instance Methods
     public abstract byte[ ] getEncoded( ) throws CertificateEncodingException;
     public abstract java.security.PublicKey getPublicKey( );
     public final String getType( );
     public abstract void verify(java.security.PublicKey key)
        throws CertificateException, java.security.NoSuchAlgorithmException,
        java.security.InvalidKeyException, java.security.NoSuchProviderException, java.security.SignatureException;
     public abstract void verify(java.security.PublicKey key, String sigProvider)
        throws CertificateException, java.security.NoSuchAlgorithmException,
        java.security.InvalidKeyException, java.security.NoSuchProviderException, java.security.SignatureException;
// Public Methods Overriding Object
     public boolean equals(Object other);
     public int hashCode( );
     public abstract String toString( );
// Protected Instance Methods
1.3  protected Object writeReplace( ) throws java.io.ObjectStreamException;
}
```

Subclasses

`X509Certificate`

Passed To

Too many methods to list.

Returned By

Too many methods to list.

Certificate.CertificateRep java.security.cert

Java 1.3 ***serializable***

This protected inner class provides an alternate representation of a certificate that can be used for serialization purposes by the `writeReplace( )` method of some `Certificate` implementations. Applications do not typically need this class.

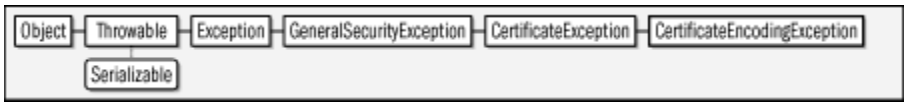
```
protected static class Certificate.CertificateRep implements Serializable {
// Protected Constructors
     protected CertificateRep(String type, byte[ ] data);
// Protected Instance Methods
     protected Object readResolve( ) throws java.io.ObjectStreamException;
}
```

CertificateEncodingException java.security.cert

Java 1.2 ***serializable checked***

Signals an error while attempting to encode a certificate.

Figure 14-48. java.security.cert.CertificateEncodingException




```
public class CertificateEncodingException extends CertificateException {
// Public Constructors
     public CertificateEncodingException( );
5.0  public CertificateEncodingException(Throwable cause);
     public CertificateEncodingException(String message);
5.0  public CertificateEncodingException(String message, Throwable cause);
}
```

Thrown By

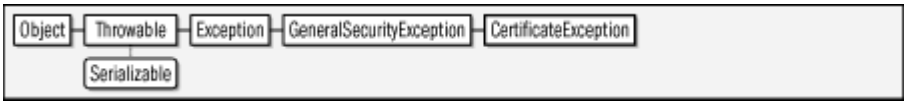
`java.security.cert.Certificate.getEncoded( )`, `CertPath.getEncoded( )`, `X509Certificate.getTBSCertificate( )`

CertificateException **java.security.cert**

Java 1.2 ***serializable checked***

This class is the superclass of several more specific exception types that may be thrown when working with certificates.

Figure 14-49. java.security.cert.CertificateException




```
public class CertificateException
       extends java.security.GeneralSecurityException {
// Public Constructors
     public CertificateException( );
5.0  public CertificateException(Throwable cause);
     public CertificateException(String msg);
5.0  public CertificateException(String message, Throwable cause);
}
```

Subclasses

`CertificateEncodingException`, `CertificateExpiredException`, `CertificateNotYetValidException`, `CertificateParsingException`

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

Thrown By

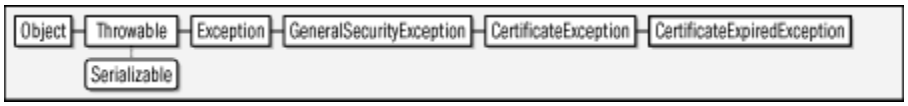
Too many methods to list.

CertificateExpiredException java.security.cert

Java 1.2 ***serializable checked***

Signals that a certificate has expired or will have expired by a specified date.

Figure 14-50. java.security.cert.CertificateExpiredException




```
public class CertificateExpiredException extends CertificateException {
// Public Constructors
     public CertificateExpiredException( );
     public CertificateExpiredException(String message);
}
```

Thrown By

`X509Certificate.checkValidity( )`

CertificateFactory java.security.cert

Java 1.2

This class defines methods for parsing certificates, certificate chains (certification paths) and certificate revocation lists (CRLs) from byte streams. Obtain a `CertificateFactory` by calling one of the static `getInstance( )` factory methods and specifying the type of certificate or CRL to be parsed, and, optionally, the desired service provider to perform the parsing. The default "SUN" provider defines only a single "X.509" certificate type, so you typically obtain a `CertificateFactory` with this code:

```
CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
```

Once you have obtained a `CertificateFactory` for the desired type of certificate, call `generateCertificate( )` to parse a `Certificate` from a specified byte stream, or call `generateCertificates( )` to parse a group of unrelated certificates (i.e. certificates that do not form a certificate chain) from a stream and return them as a `Collection` of `Certificate` objects. Similarly, call `generateCRL( )` to parse a single

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

CRL object from a stream, and call `generateCRLs( )` to parse a `Collection` of `CRL` objects from the stream. These `CertificateFactory` methods read to the end of the specified stream. If the stream supports `mark( )` and `reset( )`, however, the `CertificateFactory` resets the stream to the position after the end of the last certificate or CRL read. If you specified a certificate type of "X.509", the `Certificate` and `CRL` objects returned by a `CertificateFactory` can be cast safely to `X509Certificate` and `X509CRL`. A certificate factory for X.509 certificates can parse certificates encoded in binary or printable hexadecimal form. If the certificate is in hexadecimal form, it must begin with the string "——-BEGIN CERTIFICATE——-" and end with the string "——-END CERTIFICATE——-".

The `generateCertPath( )` methods return a `CertPath` object representing a certificate chain. These methods can create a `CertPath` object from a `List` of `Certificate` object, or by reading the chained certificates from a stream. Specify the encoding of the certificate chain by passing the name of the encoding standard to `generateCertPath( )`. The default "SUN" provider supports the "PKCS7" and the "PkiPath" encodings. `getCertPathEncoding( )` returns an `Iterator` of the encodings supported by the current provider. The first encoding returned by the iterator is the default used when no encoding is explicitly specified.

```
public class CertificateFactory {
// Protected Constructors
     protected CertificateFactory(CertificateFactorySpi certFacSpi,
        java.security.Provider provider, String type);
// Public Class Methods
     public static final CertificateFactory getInstance(String type)
        throws CertificateException;
1.4  public static final CertificateFactory getInstance(String type,
        java.security.Provider provider)
        throws CertificateException;
     public static final CertificateFactory getInstance(String type,
        String provider)
        throws CertificateException, java.security.NoSuchProviderException;
// Public Instance Methods
     public final java.security.cert.Certificate generateCertificate
        (java.io.InputStream inStream)
        throws CertificateException;
     public final java.util.Collection<? extends java.security.cert.Certificate>
        generateCertificates(java.io.InputStream inStream)
        throws CertificateException;
1.4  public final CertPath generateCertPath(java.util.List<?
        extends java.security.cert.Certificate> certificates)
        throws CertificateException;
1.4  public final CertPath generateCertPath(java.io.InputStream inStream)
        throws CertificateException;
1.4  public final CertPath generateCertPath(java.io.InputStream inStream,
        String encoding)
        throws CertificateException;
     public final CRL generateCRL(java.io.InputStream inStream)
        throws CRLException;
     public final java.util.Collection<? extends CRL> generateCRLs
        (java.io.InputStream inStream)
        throws CRLException;
1.4  public final java.util.Iterator<String> getCertPathEncodings( );
     public final java.security.Provider getProvider( );
     public final String getType( );
}
```


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

CertificateFactorySpi java.security.cert

Java 1.2

This abstract class defines the service provider interface, or SPI, for the `CertificateFactory` class. A security provider must implement this class for each type of certificate it wishes to support. Applications never need to use or subclass this class.

```
public abstract class CertificateFactorySpi {
// Public Constructors
     public CertificateFactorySpi( );
// Public Instance Methods
     public abstract java.security.cert.Certificate engineGenerateCertificate
        (java.io.InputStream inStream)
        throws CertificateException;
     public abstract java.util.Collection<? extends java.security.cert.Certificate>
        engineGenerateCertificates(java.io.InputStream inStream)
        throws CertificateException;
1.4  public CertPath engineGenerateCertPath(java.util.List<?
        extends java.security.cert.Certificate> certificates)
        throws CertificateException;
1.4  public CertPath engineGenerateCertPath(java.io.InputStream inStream)
        throws CertificateException;
1.4  public CertPath engineGenerateCertPath(java.io.InputStream inStream,
        String encoding) throws CertificateException;
     public abstract CRL engineGenerateCRL(java.io.InputStream inStream)
        throws CRLException;
     public abstract java.util.Collection<? extends CRL> engineGenerateCRLs
        (java.io.InputStream inStream)
        throws CRLException;
1.4  public java.util.Iterator<String> engineGetCertPathEncodings( );
}
```

Passed To

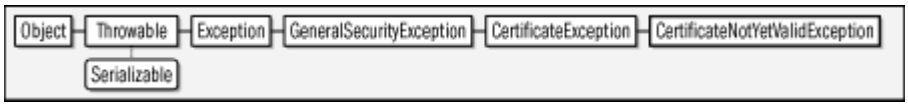
`CertificateFactory.CertificateFactory( )`

CertificateNotYetValidException java.security.cert

Java 1.2 ***serializable checked***

Signals that a certificate is not yet valid or will not yet be valid on a specified date.

Figure 14-51. java.security.cert.CertificateNotYetValidException



```
public class CertificateNotYetValidException extends CertificateException {
// Public Constructors
     public CertificateNotYetValidException( );
     public CertificateNotYetValidException(String message);
}
```

Thrown By

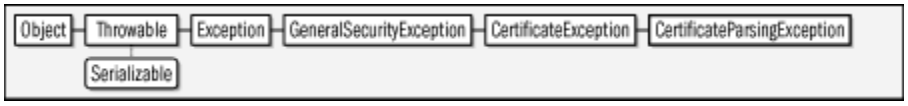
`X509Certificate.checkValidity( )`

CertificateParsingException java.security.cert

Java 1.2 ***serializable checked***

Signals an error or other problem while parsing a certificate.

Figure 14-52. java.security.cert.CertificateParsingException




```
public class CertificateParsingException extends CertificateException {
// Public Constructors
     public CertificateParsingException( );
5.0  public CertificateParsingException(Throwable cause);
     public CertificateParsingException(String message);
5.0  public CertificateParsingException(String message, Throwable cause);
}
```

Thrown By

`X509Certificate.{getExtendedKeyUsage( ), getIssuerAlternativeNames( ),getSubjectAlternativeNames( )}`

CertPath java.security.cert

Java 1.4 ***serializable***

A `CertPath` is a immutable sequence or chain of certificates that establishes a "certification path" from an unknown "end entity" to a known and trusted Certificate Authority or "trust anchor". Use a `CertPathValidator` to validate a certificate chain and establish trust in the public key presented in the certificate of the end entity.

`getType( )` returns the type of the certificates in the `CertPath`. For X.509 certificate chains (the only type supported by the default "SUN" provider) this method returns "X.

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

509". `getCertificates( )` returns a `java.util.List` object that contains the `Certificate` objects that comprise the chain. For X.509 chains, the list contains `X509Certificate` objects. Also, for X.509 certificate paths, the `List` returned by `getCertificates( )` starts with the certificate of of the end entity, and ends with a certificate signed by the trust anchor. The signer of any certificate but the last must be the subject of the next certificate in the `List`. If the end entity presents a certificate that is directly signed by a trust anchor (which is a not uncommon occurrence) then the `List` returned by `getCertificates( )` consists of only that single certificate. Note that the list of certificates does not include the certificate of the trust anchor. The public keys of trusted CAs must be known by the system in advance. In Sun's JDK implementation, the public-key certificates of trusted CAs are stored in the file *`jre/lib/security/cacerts`*.

`CertPath` objects can be created with a `CertificateFactory`, or at a lower level with a `CertPathBuilder` object. A `CertificateFactory` can parse or decode a `CertPath` object from a binary stream. The `getEncoded( )` methods reverse the process and encode a `CertPath` into an array of bytes. `getEncodings( )` returns the encodings supported for a `CertPath`. The first returned encoding name is the default one, but you can use any supported encoding by using the one-argument version of `getEncoded( )`. The default "SUN" provider supports encodings named "PKCS7" and "PkiPath".

`CertPath` objects are immutable as is the `List` object returned by `getCertificates( )` and the `Certificate` objects contained in the list. Furthermore, all `CertPath` methods are threadsafe.

Figure 14-53. java.security.cert.CertPath


```
public abstract class CertPath implements Serializable {
// Protected Constructors
     protected CertPath(String type);
// Nested Types
     protected static class CertPathRep implements Serializable;
// Public Instance Methods
     public abstract java.util.List<? extends java.security.cert.Certificate>
        getCertificates( );
     public abstract byte[ ] getEncoded( ) throws CertificateEncodingException;
     public abstract byte[ ] getEncoded(String encoding)
        throws CertificateEncodingException;
     public abstract java.util.Iterator<String> getEncodings( );
     public String getType( );
// Public Methods Overriding Object
     public boolean equals(Object other);
     public int hashCode( );
     public String toString( );
// Protected Instance Methods
     protected Object writeReplace( ) throws java.io.ObjectStreamException;
}
```


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

Passed To

`java.security.CodeSigner.CodeSigner( )`,
`java.security.Timestamp.Timestamp( )`,
`CertPathValidator.validate( )`,
`CertPathValidatorException.CertPathValidatorException( )`,
`CertPathValidatorSpi.engineValidate( )`,
`PKIXCertPathBuilderResult.PKIXCertPathBuilderResult( )`

Returned By

`java.security.CodeSigner.getSignerCertPath( )`,
`java.security.Timestamp.getSignerCertPath( )`,
`CertificateFactory.generateCertPath( )`,
`CertificateFactorySpi.engineGenerateCertPath( )`,
`CertPathBuilderResult.getCertPath( )`,
`CertPathValidatorException.getCertPath( )`,
`PKIXCertPathBuilderResult.getCertPath( )`

CertPath.CertPathRep java.security.cert

Java 1.4 ***serializable***

This protected inner class defines an implementation-independent representation of a `CertPath` for serialization purposes. Applications never need to use this class.

```
protected static class CertPath.CertPathRep implements Serializable {
// Protected Constructors
     protected CertPathRep(String type, byte[ ] data);
// Protected Instance Methods
     protected Object readResolve( ) throws java.io.ObjectStreamException;
}
```

CertPathBuilder java.security.cert

Java 1.4

`CertPathBuilder` attempts to build a certification path from a specified certificate to a trust anchor. Unlike the `CertificateFactory.generateCertPath( )` method, which might be used by a server to parse a certificate chain presented to it by a client, this class is used to create a new certificate chain, and might be used by a client that needs to send a certificate chain to a server. The `CertPathBuilder` API is provider-based, and is

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

algorithm independent, although the use of any algorithms other than the "PKIX" standards (which work with X.509 certificate chains) require appropriate external implementations of `CertPathParameters` and `CertPathBuilderResult`.

Obtain a `CertPathBuilder` object by calling one of the static `getInstance( )` methods, specifying the desired algorithm and, optionally, the desired provider. The "PKIX" algorithm is the only one supported by the default "SUN" provider, and is the only one that has the required algorithm-specific classes defined by this package. Once you have a `CertPathBuilder`, you create a `CertPath` object by passing a `CertPathParameters` object to the `build( )` method. `CertPathParameters` is a marker interfaces that defines no method of its own, so you must use an algorithm-specific implementation such as `PKIXBuilderParameters` to supply the information required to build a `CertPath`. The `build( )` method returns a `CertPathBuilderResult` object. Use the `getCertPath( )` method of this returned object to obtain the `CertPath` that was built. The algorithm-specific implementation `PKIXCertPathBuilderResult` has additional methods that return further algorithm-specific results.

```
public class CertPathBuilder {
// Protected Constructors
     protected CertPathBuilder(CertPathBuilderSpi builderSpi,
        java.security.Provider provider, String algorithm);
// Public Class Methods
     public static final String getDefaultType( );
     public static CertPathBuilder getInstance(String algorithm)
        throws java.security.NoSuchAlgorithmException;
     public static CertPathBuilder getInstance(String algorithm, String provider)
        throws java.security.NoSuchAlgorithmException,
        java.security.NoSuchProviderException;
     public static CertPathBuilder getInstance(String algorithm,
        java.security.Provider provider)
        throws java.security.NoSuchAlgorithmException;
// Public Instance Methods
     public final CertPathBuilderResult build(CertPathParameters params)
        throws CertPathBuilderException,
        java.security.InvalidAlgorithmParameterException;
     public final String getAlgorithm( );
     public final java.security.Provider getProvider( );
}
```

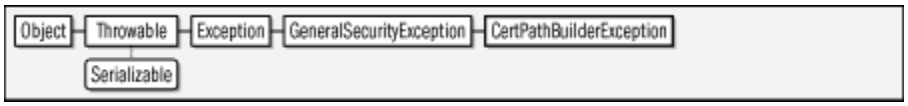
CertPathBuilderException **java.security.cert**

Java 1.4 ***serializable checked***

Signal a problem while building a certification path with `CertPathBuilder`.

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

Figure 14-54. java.security.cert.CertPathBuilderException




```
public class CertPathBuilderException
        extends java.security.GeneralSecurityException {
// Public Constructors
     public CertPathBuilderException( );
     public CertPathBuilderException(Throwable cause);
     public CertPathBuilderException(String msg);
     public CertPathBuilderException(String msg, Throwable cause);
}
```

Thrown By

`CertPathBuilder.build( )`,`CertPathBuilderSpi.engineBuild( )`

CertPathBuilderResult java.security.cert

Java 1.4 ***cloneable***

An object of this type is returned by the `build( )` method of a `CertPathBuilder`. The `getCertPath( )` method returns the `CertPath` object that was built; this method will never return `null`. The algorithm-specific `PKIXCertPathBuilderResult` implementation defines other methods to return additional information about the path that was built.

Figure 14-55. java.security.cert.CertPathBuilderResult


```
public interface CertPathBuilderResult extends Cloneable {
// Public Instance Methods
     Object clone( );
     CertPath getCertPath( );
}
```

Implementations

`PKIXCertPathBuilderResult`

Returned By

`CertPathBuilder.build( )`,`CertPathBuilderSpi.engineBuild( )`

CertPathBuilderSpi java.security.cert

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

Java 1.4

This abstract class defines the Service Provider Interface for the `CertPathBuilder`. Security providers must implement this interface, but applications never need to use it.

```
public abstract class CertPathBuilderSpi {
// Public Constructors
     public CertPathBuilderSpi( );
// Public Instance Methods
     public abstract CertPathBuilderResult engineBuild(CertPathParameters params)
        throws CertPathBuilderException,
        java.security.InvalidAlgorithmParameterException;
}
```

Passed To

`CertPathBuilder.CertPathBuilder( )`

CertPathParameters java.security.cert

Java 1.4 ***cloneable***

`CertPathParamters` is a marker interface for objects that hold parameters (such as the set of trust anchors) for validating or building a certification path with `CertPathValidator` and `CertPathBuilder`. It defines no methods of its own, but requires that all implementations include a working `clone( )` method. You must use an algorithm-specific implementation of this interface, such as `PKIXParameters` or `PKIXBuilderParameters` when validating or building a `CertPath`, and it is rarely useful to work with this interface directly.

Figure 14-56. java.security.cert.CertPathParameters


```
public interface CertPathParameters extends Cloneable {
// Public Instance Methods
     Object clone( );
}
```

Implementations

`PKIXParameters`

Passed To

`CertPathBuilder.build( ),CertPathBuilderSpi.engineBuild( ), CertPathValidator.validate( ), CertPathValidatorSpi.engineValidate( ),`

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

`javax.net.ssl.CertPathTrustManagerParameters.CertPathTrustManagerParameters( )`

Returned By

`javax.net.ssl.CertPathTrustManagerParameters.getParameters( )`

CertPathValidator java.security.cert

Java 1.4

This class validates certificate chains, establishing a chain of trust from the end entity to a trust anchor, and thereby establishing the validity of the public key presented in the end entity's certificate. The `CertPathValidator` is provider-based and algorithm-independent. To obtain a `CertPathValidator` instance, call one of the static `getInstance( )` methods specifying the name of the desired validation algorithm and, optionally, the provider to use. The "PKIX" algorithm for validating X.509 certificates is the only one supported by the default "SUN" provider.

Once you have a `CertPathValidator` object, you can use it to validate certificate chains by passing the `CertPath` object to be validated to the `validate( )` method along with a `CertPathParameters` object that specifies valid trust anchors and other validation parameters. `CertPathParameters` is simply a marker interface, and you must use an application-specific implementation such as `PKIXParameters`. If validation fails, the `validate( )` method throws a `CertPathValidatorException` which may include the index in the chain of the certificate that failed to validate. Otherwise, if validation is successful, the `validate( )` method returns a `CertPathValidatorResult`. If you are interested in the details of the validation (such as the trust anchor that was used or the public key of the end entity), you may cast this returned value to an algorithm-specific subtype such as `PKIXCertPathValidatorResult` and use its methods to find out more about the result.

```
public class CertPathValidator {
// Protected Constructors
     protected CertPathValidator(CertPathValidatorSpi validatorSpi,
        java.security.Provider provider, String algorithm);
// Public Class Methods
     public static final String getDefaultType( );
     public static CertPathValidator getInstance(String algorithm)
        throws java.security.NoSuchAlgorithmException;
     public static CertPathValidator getInstance(String algorithm,
        String provider)
        throws java.security.NoSuchAlgorithmException,
        java.security.NoSuchProviderException;
     public static CertPathValidator getInstance(String algorithm,
        java.security.Provider provider)
        throws java.security.NoSuchAlgorithmException;
// Public Instance Methods
     public final String getAlgorithm( );
     public final java.security.Provider getProvider( );
     public final CertPathValidatorResult validate(CertPath certPath,
```


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

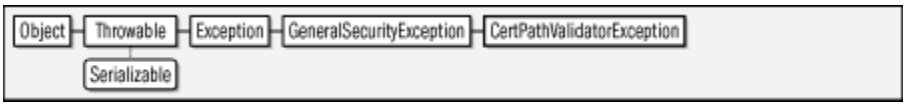
```
        CertPathParameters params)
        throws CertPathValidatorException,
        java.security.InvalidAlgorithmParameterException;
}
```

CertPathValidatorException java.security.cert

Java 1.4 ***serializable checked***

Signals a problem while validating a certificate chain with a `CertPathValidator`. `getCertPath( )` returns the `CertPath` object that was being validated, and `getIndex( )` returns the index within the path of the certificate that caused the exception (or -1 if that information is not available).

Figure 14-57. java.security.cert.CertPathValidatorException




```
public class CertPathValidatorException
        extends java.security.GeneralSecurityException {
// Public Constructors
     public CertPathValidatorException( );
     public CertPathValidatorException(Throwable cause);
     public CertPathValidatorException(String msg);
     public CertPathValidatorException(String msg, Throwable cause);
     public CertPathValidatorException(String msg, Throwable cause,
        CertPath certPath, int index);
// Public Instance Methods
     public CertPath getCertPath( );                      default:null
     public int getIndex( );                              default:-1
}
```

Thrown By

`CertPathValidator.validate( )`, `CertPathValidatorSpi.engineValidate( )`, `PKIXCertPathChecker.{check( ),init( )}`

CertPathValidatorResult java.security.cert

Java 1.4 ***cloneable***

This marker interface defines the type of the object returned by the `validate( )` method of a `CertPathValidator`, but does not define any of the contents of that object, other

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

to specify that it must be `Cloneable`. If you want any details about the results of validating a `CertPath`, you must cast the return value of `validate( )` to an algorithm-specific types implementation of this interface, such as `PKIXCertPathValidatorResult`.

Figure 14-58. java.security.cert.CertPathValidatorResult


```
public interface CertPathValidatorResult extends Cloneable {
// Public Instance Methods
     Object clone( );
}
```

Implementations

`PKIXCertPathValidatorResult`

Returned By

`CertPathValidator.validate( ), CertPathValidatorSpi.engineValidate( )`

CertPathValidatorSpi java.security.cert

Java 1.4

This abstract class defines the Service Provider Interface for the `CertPathValidator` class. Security providers must implement this interface, but applications never need to use it.

```
public abstract class CertPathValidatorSpi {
// Public Constructors
     public CertPathValidatorSpi( );
// Public Instance Methods
     public abstract CertPathValidatorResult engineValidate(CertPath certPath,
        CertPathParameters params)
        throws CertPathValidatorException,
        java.security.InvalidAlgorithmParameterException;
}
```

Passed To

`CertPathValidator.CertPathValidator( )`

CertSelector java.security.cert

Java 1.4 ***cloneable***

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

This interface defines an API for determining whether a `Certificate` meets some criteria. Implementations are used to specify critera by which a certificate or certificates should be selected from a `CertStore` object. The `match( )` method should examine the `Certificate` it is passed and return `true` if it "matches" based on whatever criteria the implementation defines. See `X509CertSelector` for an implementation that works with X.509 certificates. See `CRLSelector` for a similar interface for use when selecting `CRL` objects from a `CertStore`.

Figure 14-59. java.security.cert.CertSelector


```
public interface CertSelector extends Cloneable {
// Public Instance Methods
     Object clone( );
     boolean match(java.security.cert.Certificate cert);
}
```

Implementations

`X509CertSelector`

Passed To

`CertStore.getCertificates( )`,
`CertStoreSpi.engineGetCertificates( )`,
`PKIXBuilderParameters.PKIXBuilderParameters( )`,
`PKIXParameters.setTargetCertConstraints( )`

Returned By

`PKIXParameters.getTargetCertConstraints( )`

CertStore java.security.cert

Java 1.4

A `CertStore` object is a repository for `Certificate` and `CRL` objects. You may query a `CertStore` for a `java.util.Collection` of `Certificate` or `CRL` objects that match specified criteria by passing a `CertSelector` or `CRLSelector` to `getCertificates( )` or `getCRLs( )`. A `CertStore` is conceptually similar to a `java.security.KeyStore`, but there are significant differences in how the two classes are intended to be used. A `KeyStore` is designed to store a relatively small local collection of private keys and trusted certificates. A `CertStore`, however, may represent a large public database (in the form of an LDAP server, for examle) of untrusted certificates.

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

Obtain a `CertStore` object by calling a `getInstance( )` method and specifying the name of the desired `CertStore` type and a `CertStoreParameters` object that is specific to that type. Optionally, you may also specify the desired provider of your `CertStore` object. The default "SUN" provider defines two `CertStore` types, named "LDAP" and "Collection", which you should use with `LDAPCertStoreParameters` and `CollectionCertStoreParameters` objects, respectively. The "LDAP" type obtains certificates and CRLs from a network LDAP server, and the "Collection" type obtains them from a a specified `Collection` object.

The `CertStore` class may be directly useful to applications that want to query a LDAP server for certificates. It is also used by `PKIXParameters.addCertStore( )` and `PKIXParameters.setCertStores( )` to specify a source of certificates to by used by the `CertPathBuilder` and `CertPathValidator` classes.

All public methods of `CertStore` are threadsafe.

```
public class CertStore {
// Protected Constructors
     protected CertStore(CertStoreSpi storeSpi, java.security.Provider provider,
        String type, CertStoreParameters params);
// Public Class Methods
     public static final String getDefaultType( );
     public static CertStore getInstance(String type, CertStoreParameters params)
        throws java.security.InvalidAlgorithmParameterException,
        java.security.NoSuchAlgorithmException;
     public static CertStore getInstance(String type, CertStoreParameters params,
        String provider)
        throws java.security.InvalidAlgorithmParameterException,
        java.security.NoSuchAlgorithmException,
        java.security.NoSuchProviderException;
     public static CertStore getInstance(String type, CertStoreParameters params,
        java.security.Provider provider)
        throws java.security.NoSuchAlgorithmException,
        java.security.InvalidAlgorithmParameterException;
// Public Instance Methods
     public final java.util.Collection<? extends java.security.cert.Certificate>
        getCertificates(CertSelector selector)
        throws CertStoreException;
     public final CertStoreParameters getCertStoreParameters( );
     public final java.util.Collection<? extends CRL> getCRLs
        (CRLSelector selector)
        throws CertStoreException;
     public final java.security.Provider getProvider( );
     public final String getType( );
}
```

Passed To

`PKIXParameters.addCertStore( )`

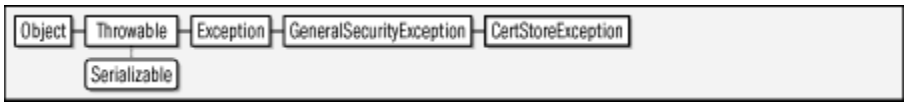
CertStoreException **java.security.cert**

Java 1.4 ***serializable checked***

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

Signals a problem while querying a `CertStore` for certificates or CRLs.

Figure 14-60. java.security.cert.CertStoreException




```
public class CertStoreException extends java.security.GeneralSecurityException {
// Public Constructors
     public CertStoreException( );
     public CertStoreException(Throwable cause);
     public CertStoreException(String msg);
     public CertStoreException(String msg, Throwable cause);
}
```

Thrown By

`CertStore.{getCertificates( ),getCRLs( )},CertStoreSpi.{engineGetCertificates( ),engineGetCRLs( )}`

CertStoreParameters java.security.cert

Java 1.4 ***cloneable***

This marker interface defines the type, but not the content, of the parameters object that is passed to the `CertStore.getInstance( )` methods. It does not define any methods of its own and simply requires that all implementing classes be cloneable. Use one of the concrete implementations of this class for `CertStore` objects of type "LDAP" and "Collection".

Figure 14-61. java.security.cert.CertStoreParameters


```
public interface CertStoreParameters extends Cloneable {
// Public Instance Methods
     Object clone( );
}
```

Implementations

`CollectionCertStoreParameters, LDAPCertStoreParameters`

Passed To

`CertStore.{CertStore( ),getInstance( )}, CertStoreSpi.CertStoreSpi( )`

Returned By

`CertStore.getCertStoreParameters( )`

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

CertStoreSpi java.security.cert

Java 1.4

This abstract class defines the Service Provider Interface for the `CertStore` class. Security providers must implement this interface, but applications never need to use it.

```
public abstract class CertStoreSpi {
// Public Constructors
     public CertStoreSpi(CertStoreParameters params)
        throws java.security.InvalidAlgorithmParameterException;
// Public Instance Methods
     public abstract java.util.Collection<? extends java.security.cert.
        Certificate> engineGetCertificates(CertSelector selector)
        throws CertStoreException;
     public abstract java.util.Collection<? extends CRL>
        engineGetCRLs(CRLSelector selector)
        throws CertStoreException;
}
```

Passed To

`CertStore.CertStore( )`

CollectionCertStoreParameters java.security.cert

Java 1.4 ***cloneable***

This concrete implementation of `CertStoreParameters` is used when creating a `CertStore` object of type "Collection". Pass the `Collection` of `Certificate` and `CRL` objects to be searched by the `CertStore` to the constructor method.

Figure 14-62. java.security.cert.CollectionCertStoreParameters


```
public class CollectionCertStoreParameters implements CertStoreParameters {
// Public Constructors
     public CollectionCertStoreParameters( );
     public CollectionCertStoreParameters(java.util.Collection<?> collection);
// Public Instance Methods
     public java.util.Collection<?> getCollection( );
// Methods Implementing CertStoreParameters
     public Object clone( );
// 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

CRL java.security.cert

Java 1.2

This abstract class represents a *certificate revocation list* (CRL). A CRL is an object issued by a certificate authority (or other certificate signer) that lists certificates that have been revoked, meaning that they are now invalid and should be rejected. Use a `CertificateFactory` to parse a `CRL` from a byte stream. Use the `isRevoked( )` method to test whether a specified `Certificate` is listed on the `CRL`. Note that type-specific `CRL` subclasses, such as `X509CRL`, may provide access to substantially more information about the revocation list.

```
public abstract class CRL {
// Protected Constructors
     protected CRL(String type);
// Public Instance Methods
     public final String getType( );
     public abstract boolean isRevoked(java.security.cert.Certificate cert);
// Public Methods Overriding Object
     public abstract String toString( );
}
```

Subclasses

`X509CRL`

Passed To

`CRLSelector.match( ),X509CRLSelector.match( )`

Returned By

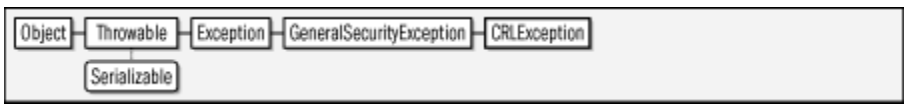
`CertificateFactory.generateCRL( ), CertificateFactorySpi.engineGenerateCRL( )`

CRLException java.security.cert

Java 1.2 ***serializable checked***

Signals an error or other problem while working with a CRL.

Figure 14-63. java.security.cert.CRLException



```
public class CRLException extends java.security.GeneralSecurityException {
// Public Constructors
     public CRLException( );
5.0  public CRLException(Throwable cause);
     public CRLException(String message);
5.0  public CRLException(String message, Throwable cause);
}
```

Thrown By

`CertificateFactory.{generateCRL( ),generateCRLs( )}, CertificateFactorySpi.{engineGenerateCRL( ), engineGenerateCRLs( )},X509CRL.{getEncoded( ),getTBSCertList( ), verify( )},X509CRLEntry.getEncoded( )`

CRLSelector java.security.cert

Java 1.4 ***cloneable***

This interface defines an API for determining whether a `CRL` object meets some criteria. Implementations are used to specify critera by which a `CRL` objects should be selected from a `CertStore`. The `match( )` method should examine the `CRL` it is passed and return `true` if it "matches" based on whatever criteria the implementation defines. See `X509CRLSelector` for an implementation that works with X.509 certificates. See `CertSelector` for a similar interface for use when selecting `Certificate` objects from a `CertStore`.

Figure 14-64. java.security.cert.CRLSelector


```
public interface CRLSelector extends Cloneable {
// Public Instance Methods
     Object clone( );
     boolean match(CRL crl);
}
```

Implementations

`X509CRLSelector`

Passed To

`CertStore.getCRLs( ),CertStoreSpi.engineGetCRLs( )`

LDAPCertStoreParameters java.security.cert

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

Java 1.4 ***cloneable***

This concrete implementation of `CertStoreParameters` is used when creating a `CertStore` object of type "LDAP". It specifies the hostname of the LDAP server to connect to and, optionally, the port to connect on.

Figure 14-65. java.security.cert.LDAPCertStoreParameters


```
public class LDAPCertStoreParameters implements CertStoreParameters {
// Public Constructors
     public LDAPCertStoreParameters( );
     public LDAPCertStoreParameters(String serverName);
     public LDAPCertStoreParameters(String serverName, int port);
// Public Instance Methods
     public int getPort( );                          default:389
     public String getServerName( );                 default:"localhost"
// Methods Implementing CertStoreParameters
     public Object clone( );
// Public Methods Overriding Object
     public String toString( );
}
```

PKIXBuilderParameters **java.security.cert**

Java 1.4 ***cloneable***

Instances of this class are used to specify parameters to the `build( )` method of a `CertPathBuilder` object. These parameters must include the two mandatory ones passed to the constructors. The first is a source of trust anchors, which may be supplied as a `Set` of `TrustAnchor` objects or as a `java.security.KeyStore` object. The second required parameter is a `CertSelector` object (typically an `X509CertSelector`) that specifies the selection criteria for the certificate that is to have the certification path built. In addition to these parameters that are passed to the constructor, this class also inherits a number of methods for setting other parameters, and defines `setMaxPathLength( )` for specifying the maximum length of the certificate chain that is built.

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

Figure 14-66. java.security.cert.PKIXBuilderParameters

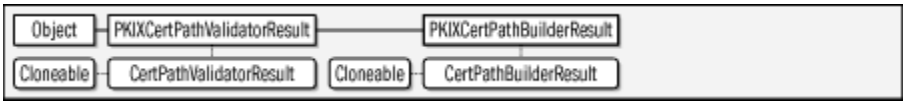

```
public class PKIXBuilderParameters extends PKIXParameters {
// Public Constructors
     public PKIXBuilderParameters(java.security.KeyStore keystore,
        CertSelector targetConstraints)
        throws java.security.KeyStoreException,
        java.security.InvalidAlgorithmParameterException;
     public PKIXBuilderParameters(java.util.Set<TrustAnchor> trustAnchors,
        CertSelector targetConstraints)
        throws java.security.InvalidAlgorithmParameterException;
// Public Instance Methods
     public int getMaxPathLength( );
     public void setMaxPathLength(int maxPathLength);
// Public Methods Overriding PKIXParameters
     public String toString( );
}
```

PKIXCertPathBuilderResult java.security.cert

Java 1.4 ***cloneable***

An instance of this class is retured by the `build( )` method of a `CertPathBuilder` created for the "PKIX" algorithm. `getCertPath( )` returns the `CertPath` object that was built, and methods inherited from the superclass return additional information such as the public key of the subject of the certificate chain and the trust anchor that terminates the chain.

Figure 14-67. java.security.cert.PKIXCertPathBuilderResult




```
public class PKIXCertPathBuilderResult extends PKIXCertPathValidatorResult
     implements CertPathBuilderResult {
// Public Constructors
     public PKIXCertPathBuilderResult(CertPath certPath, TrustAnchor trustAnchor,
        PolicyNode policyTree,
        java.security.PublicKey subjectPublicKey);
// Methods Implementing CertPathBuilderResult
     public CertPath getCertPath( );
// Public Methods Overriding PKIXCertPathValidatorResult
     public String toString( );
}
```

PKIXCertPathChecker java.security.cert

Java 1.4 ***cloneable***

This abstract class defines an extension mechanism for the PKIX certification path building and validation algorithms. Most applications will never need to use this class. You may pass one or more `PKIXCertPathChecker` objects to the `setCertPathCheckers( )` or `addCertPathChecker( )` methods of the `PKIXParameters` or `PKIXBuilderParameters` object that is passed to the `build( )` or `validate( )` methods of a `CertPathBuilder` or `CertPathValidator`. The `check( )` method of all `PKIXCertPathChecker` objects registered in this way will be invoked for each certificate considered in the building or validation algorithms. `check( )` should throw a `CertPathValidatorException` if a certificate does not the implemented test. The `init( )` method is invoked to tell the checker to reset its internal state and to notify it of the direction in which certificates will be presented. Checkers are not required to support the forward direction, and should return `false` from `isForwardCheckingSupported( )` if they do not.

Figure 14-68. java.security.cert.PKIXCertPathChecker


```
public abstract class PKIXCertPathChecker implements Cloneable {
// Protected Constructors
     protected PKIXCertPathChecker( );
// Public Instance Methods
     public abstract void check(java.security.cert.Certificate cert,
        java.util.Collection<String> unresolvedCritExts)
        throws CertPathValidatorException;
     public abstract java.util.Set<String> getSupportedExtensions( );
     public abstract void init(boolean forward) throws CertPathValidatorException;
     public abstract boolean isForwardCheckingSupported( );
// Public Methods Overriding Object
     public Object clone( );
}
```

Passed To

`PKIXParameters.addCertPathChecker( )`

PKIXCertPathValidatorResult java.security.cert

Java 1.4 ***cloneable***

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

An instance of this class is returned upon successful validation by the `validate( )` method of a `CertPathValidator` created for the "PKIX" algorithm. `getPublicKey( )` returns the validated public key of the subject of the certificate chain. `getTrustAnchor( )` returns the `TrustAnchor` that anchors the chain.

Figure 14-69. java.security.cert.PKIXCertPathValidatorResult


```
public class PKIXCertPathValidatorResult implements CertPathValidatorResult {
// Public Constructors
     public PKIXCertPathValidatorResult(TrustAnchor trustAnchor,
        PolicyNode policyTree,
        java.security.PublicKey subjectPublicKey);
// Public Instance Methods
     public PolicyNode getPolicyTree( );
     public java.security.PublicKey getPublicKey( );
     public TrustAnchor getTrustAnchor( );
// Methods Implementing CertPathValidatorResult
     public Object clone( );
// Public Methods Overriding Object
     public String toString( );
}
```

Subclasses

`PKIXCertPathBuilderResult`

PKIXParameters — java.security.cert

Java 1.4 — ***cloneable***

This implementation of `CertPathParameters` defines parameters that are passed to the `validate( )` method of a PKIX `CertPathValidator` and defines a subset of the parameters that are passed to the `build( )` method of a PKIX `CertPathBuilder`. A full understanding of this class requires a detailed discussion of the PKIX certification path building and validation algorithms, which is beyond the scope of this book. However, some of the more important parameters are described here.

When you create a `PKIXParameters` object, you must specify which trust anchors are to be used. You can do this by passing a `Set` of `TrustAnchor` objects to the constructor, or by passing a `KeyStore` containing trust anchor keys to the constructor. Once a `PKIXParameters` object is created, you can modify the set of `TrustAnchor` objects with `setTrustAnchors( )`. Specify a Set of `CertStore` objects to be searched for certificates with `setCertStores( )` or add a single CertStore to the set with `addCertStore( )`.

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

If certificate validity is to be checked for some date and time other than the current time, use `setDate( )` to specify this date.

Figure 14-70. java.security.cert.PKIXParameters


```
public class PKIXParameters implements CertPathParameters {
// Public Constructors
     public PKIXParameters(java.security.KeyStore keystore)
        throws java.security.KeyStoreException,
        java.security.InvalidAlgorithmParameterException;
     public PKIXParameters(java.util.Set<TrustAnchor> trustAnchors)
throws java.security.InvalidAlgorithmParameterException;
// Public Instance Methods
     public void addCertPathChecker(PKIXCertPathChecker checker);
     public void addCertStore(CertStore store);
     public java.util.List<PKIXCertPathChecker> getCertPathCheckers( );
     public java.util.List<CertStore> getCertStores( );
     public java.util.Date getDate( );
     public java.util.Set<String> getInitialPolicies( );
     public boolean getPolicyQualifiersRejected( );
     public String getSigProvider( );
     public CertSelector getTargetCertConstraints( );
     public java.util.Set<TrustAnchor> getTrustAnchors( );
     public boolean isAnyPolicyInhibited( );
     public boolean isExplicitPolicyRequired( );
     public boolean isPolicyMappingInhibited( );
     public boolean isRevocationEnabled( );
     public void setAnyPolicyInhibited(boolean val);
     public void setCertPathCheckers(java.util.List<PKIXCertPathChecker>
        checkers);
     public void setCertStores(java.util.List<CertStore> stores);
     public void setDate(java.util.Date date);
     public void setExplicitPolicyRequired(boolean val);
     public void setInitialPolicies(java.util.Set<String> initialPolicies);
     public void setPolicyMappingInhibited(boolean val);
     public void setPolicyQualifiersRejected(boolean qualifiersRejected);
     public void setRevocationEnabled(boolean val);
     public void setSigProvider(String sigProvider);
     public void setTargetCertConstraints(CertSelector selector);
     public void setTrustAnchors(java.util.Set<TrustAnchor> trustAnchors)
throws java.security.InvalidAlgorithmParameterException;
// Methods Implementing CertPathParameters
     public Object clone( );
// Public Methods Overriding Object
     public String toString( );
}
```

Subclasses

`PKIXBuilderParameters`

PolicyNode java.security.cert

Java 1.4

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

This class represents a node in the policy tree created by the PKIX certification path validation algorithm. A discussion of X.509 policy extensions and their use in the PKIX certification path algorithms is beyond the scope of this reference.

```
public interface PolicyNode {
// Public Instance Methods
     java.util.Iterator<? extends PolicyNode> getChildren( );
     int getDepth( );
     java.util.Set<String> getExpectedPolicies( );
     PolicyNode getParent( );
     java.util.Set<? extends PolicyQualifierInfo> getPolicyQualifiers( );
     String getValidPolicy( );
     boolean isCritical( );
}
```

Passed To

`PKIXCertPathBuilderResult.PKIXCertPathBuilderResult( ), PKIXCertPathValidatorResult.PKIXCertPathValidatorResult( )`

Returned By

`PKIXCertPathValidatorResult.getPolicyTree( )`

PolicyQualifierInfo java.security.cert

Java 1.4

This class is a low-level representation of a policy qualifier information from a X.509 certificate extension. A discussion of X.509 policy extensions and their use in the PKIX certification path algorithms is beyond the scope of this reference.

```
public class PolicyQualifierInfo {
// Public Constructors
     public PolicyQualifierInfo(byte[ ] encoded) throws java.io.IOException;
// Public Instance Methods
     public final byte[ ] getEncoded( );
     public final byte[ ] getPolicyQualifier( );
     public final String getPolicyQualifierId( );
// Public Methods Overriding Object
     public String toString( );
}
```

TrustAnchor java.security.cert

Java 1.4

A `TrustAnchor` represents a certificate authority that is trusted to "anchor" a certificate chain. A `TrustAnchor` object includes the X.500 distinguished name of the CA and the

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

public key of the CA. You may specify the name and key explictly or by passing an `X509Certificate` to the `TrustAnchor( )` constructor. If you do not pass a certificate, you can specify the CA name as a `String` or as an `X500Principal` object from the `javax.security.auth.x500` package. All forms of the `TrustAnchor( )` constructor also allow you to specify a byte array containing a binary representation of a "Name Constraints" extension. The format and meaning of such name constraints is beyond the scope of this reference, and most applications can simply specify `null` for this constructor argument.

```
public class TrustAnchor {
// Public Constructors
     public TrustAnchor(X509Certificate trustedCert, byte[ ] nameConstraints);
5.0  public TrustAnchor(javax.security.auth.x500.X500Principal caPrincipal,
        java.security.PublicKey pubKey,
        byte[ ] nameConstraints);
     public TrustAnchor(String caName, java.security.PublicKey pubKey,
        byte[ ] nameConstraints);
// Public Instance Methods
5.0  public final javax.security.auth.x500.X500Principal getCA( );
     public final String getCAName( );
     public final java.security.PublicKey getCAPublicKey( );
     public final byte[ ] getNameConstraints( );
     public final X509Certificate getTrustedCert( );
// Public Methods Overriding Object
     public String toString( );
}
```

Passed To

`PKIXCertPathBuilderResult.PKIXCertPathBuilderResult( )`, `PKIXCertPathValidatorResult.PKIXCertPathValidatorResult( )`

Returned By

`PKIXCertPathValidatorResult.getTrustAnchor( )`

X509Certificate — java.security.cert

Java 1.2 — ***serializable***

This class represents an X.509 certificate. Its various methods provide complete access to the contents of the certificate. A full understanding of this class requires detailed knowledge of the X.509 standard which is beyond the scope of this reference. Some of the more important methods are described here, however. `getSubjectDN( )` returns the `Principal` to whom this certificate applies, and the inherited `getPublicKey( )` method returns the `PublicKey` that the certificate associates with that `Principal`. `getIssuerDN( )` returns a `Principal` that represents the issuer of the certificate, and if you know the public key for that `Principal`, you can pass it to the `verify( )` method to check the digital signature of the issuer and ensure that the certificate is not forged. `checkValidity( )` checks whether the certificate has expired or has not yet gone into

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

effect. Note that `verify( )` and `getPublicKey( )` are inherited from `Certificate`.

Obtain an `X509Certificate` object by creating a `CertificateFactory` for certificate type "X.509" and then using `generateCertificate( )` to parse an X.509 certificate from a stream of bytes. Finally, cast the `Certificate` returned by this method to an `X509Certificate`.

Figure 14-71. java.security.cert.X509Certificate


```
public abstract class X509Certificate extends java.security.cert.Certificate
     implements X509Extension {
// Protected Constructors
     protected X509Certificate( );
// Public Instance Methods
     public abstract void checkValidity( )
        throws CertificateExpiredException, CertificateNotYetValidException;
     public abstract void checkValidity(java.util.Date date)
        throws CertificateExpiredException, CertificateNotYetValidException;
     public abstract int getBasicConstraints( );
1.4  public java.util.List<String> getExtendedKeyUsage( )
        throws CertificateParsingException;
1.4  public java.util.Collection<java.util.List<?>> getIssuerAlternativeNames( )
        throws CertificateParsingException;
     public abstract java.security.Principal getIssuerDN( );
     public abstract boolean[ ] getIssuerUniqueID( );
1.4  public javax.security.auth.x500.X500Principal getIssuerX500Principal( );
     public abstract boolean[ ] getKeyUsage( );
     public abstract java.util.Date getNotAfter( );
     public abstract java.util.Date getNotBefore( );
     public abstract java.math.BigInteger getSerialNumber( );
     public abstract String getSigAlgName( );
     public abstract String getSigAlgOID( );
     public abstract byte[ ] getSigAlgParams( );
     public abstract byte[ ] getSignature( );
1.4  public java.util.Collection<java.util.List<?>> getSubjectAlternativeNames( )
        throws CertificateParsingException;
     public abstract java.security.Principal getSubjectDN( );
     public abstract boolean[ ] getSubjectUniqueID( );
1.4  public javax.security.auth.x500.X500Principal getSubjectX500Principal( );
     public abstract byte[ ] getTBSCertificate( )
        throws CertificateEncodingException;
     public abstract int getVersion( );
}
```

Passed To

`TrustAnchor.TrustAnchor( )`,`X509CertSelector.setCertificate( )`, `X509CRL.getRevokedCertificate( )`, `X509CRLSelector.setCertificateChecking( )`, `javax.net.ssl.X509TrustManager.{checkClientTrusted( ), checkServerTrusted( )}`, `javax.security.auth.x500.X500PrivateCredential.X500PrivateCredential( )`

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

Returned By

`TrustAnchor.getTrustedCert( )`, `X509CertSelector.getCertificate( )`, `X509CRLSelector.getCertificateChecking( )`, `javax.net.ssl.X509KeyManager.getCertificateChain( )`, `javax.net.ssl.X509TrustManager.getAcceptedIssuers( )`, `javax.security.auth.x500.X500PrivateCredential.getCertificate( )`

X509CertSelector — java.security.cert

Java 1.4 — ***cloneable***

This class is a `CertSelector` for X.509 certificates. Its various `set` methods allow you to specify values for various certificate fields and extensions. The `match( )` method will only return `true` for certificates that have the specified values for those fields and extensions. A full understanding of this class requires detailed knowledge of the X.509 standard which is beyond the scope of this reference. Some of the more important methods are described here, however.

When you want to match exactly one specific certificate, simply pass the desired `X509Certificate` to `setCertificate( )`. Constrain the subject of the certificate with `setSubject( )`, `setSubjectAlternativeNames( )`, of `addSubjectAlternativeName( )`. Constrain the issuer of the certificate with `setIssuer( )`. Constrain the public key of the certificate with `setPublicKey( )`. Constrain the certificate to be valid on a given date with `setCertificateValid( )`. And specify a specific issuer's serial number for the certificate with `setSerialNumber( )`.

Java 5.0 adds methods for identifying certificate subjects and issuers with `javax.security.auth.x500.X500Principal` objects instead of with strings.

Figure 14-72. java.security.cert.X509CertSelector


```
public class X509CertSelector implements CertSelector {
// Public Constructors
     public X509CertSelector( );
// Public Instance Methods
     public void addPathToName(int type, String name)
        throws java.io.IOException;
     public void addPathToName(int type, byte[ ] name)
        throws java.io.IOException;
     public void addSubjectAlternativeName(int type, byte[ ] name)
        throws java.io.IOException;
```


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

```
     public void addSubjectAlternativeName(int type, String name)
        throws java.io.IOException;
     public byte[ ] getAuthorityKeyIdentifier( );              default:null
     public int getBasicConstraints( );                        default:-1
     public X509Certificate getCertificate( );                 default:null
     public java.util.Date getCertificateValid( );             default:null
     public java.util.Set<String> getExtendedKeyUsage( );    default:null
5.0  public javax.security.auth.x500.X500Principal getIssuer( );  default:null
     public byte[ ] getIssuerAsBytes( )
        throws java.io.IOException;     default:null
     public String getIssuerAsString( );                       default:null
     public boolean[ ] getKeyUsage( );                         default:null
     public boolean getMatchAllSubjectAltNames( );         default:true
     public byte[ ] getNameConstraints( );                        default:null
     public java.util.Collection<java.util.List<?>>
        getPathToNames( );     default:null
     public java.util.Set<String> getPolicy( );      default:null
     public java.util.Date getPrivateKeyValid( );          default:null
     public java.math.BigInteger getSerialNumber( );       default:null
5.0  public javax.security.auth.x500.X500Principal
        getSubject( );     default:null
     public java.util.Collection<java.util.List<?>>
        getSubjectAlternativeNames( );     default:null
     public byte[ ] getSubjectAsBytes( )
        throws java.io.IOException;     default:null
     public String getSubjectAsString( );                default:null
     public byte[ ] getSubjectKeyIdentifier( );          default:null
     public java.security.PublicKey getSubjectPublicKey( );  default:null
     public String getSubjectPublicKeyAlgID( );          default:null
     public void setAuthorityKeyIdentifier(byte[ ] authorityKeyID);
     public void setBasicConstraints(int minMaxPathLen);
     public void setCertificate(X509Certificate cert);
     public void setCertificateValid(java.util.Date certValid);
     public void setExtendedKeyUsage(java.util.Set<String> keyPurposeSet)
        throws java.io.IOException;
5.0  public void setIssuer(javax.security.auth.x500.X500Principal issuer);
     public void setIssuer(byte[ ] issuerDN) throws java.io.IOException;
     public void setIssuer(String issuerDN) throws java.io.IOException;
     public void setKeyUsage(boolean[ ] keyUsage);
     public void setMatchAllSubjectAltNames(boolean matchAllNames);
     public void setNameConstraints(byte[ ] bytes) throws java.io.IOException;
     public void setPathToNames(java.util.Collection<java.util.List<?>> names)
        throws java.io.IOException;
     public void setPolicy(java.util.Set<String> certPolicySet) throws java.io.IOException;
     public void setPrivateKeyValid(java.util.Date privateKeyValid);
     public void setSerialNumber(java.math.BigInteger serial);
     public void setSubject(String subjectDN) throws java.io.IOException;
5.0  public void setSubject(javax.security.auth.x500.X500Principal subject);
     public void setSubject(byte[ ] subjectDN) throws java.io.IOException;
     public void setSubjectAlternativeNames(java.util.Collection<
        java.util.List<?>> names) throws java.io.IOException;
     public void setSubjectKeyIdentifier(byte[ ] subjectKeyID);
     public void setSubjectPublicKey(byte[ ] key) throws java.io.IOException;
     public void setSubjectPublicKey(java.security.PublicKey key);
     public void setSubjectPublicKeyAlgID(String oid) throws java.io.IOException;
// Methods Implementing CertSelector
     public Object clone( );
     public boolean match(java.security.cert.Certificate cert);
// Public Methods Overriding Object
     public String toString( );
}
```

X509CRL **java.security.cert**

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

Java 1.2

This class represents an X.509 CRL, which consists primarily of a set of `X509CRLEntry` objects. The various methods of this class provide access to the full details of the CRL, and require a complete understanding of the X.509 standard, which is beyond the scope of this reference. Use `verify( )` to check the digital signature of the CRL to ensure that it does indeed originate from the the source it specifies. Use the inherited `isRevoked( )` method to determine whether a given certificate has been revoked. If you are curious about the revocation date for a revoked certificate, obtain the `X509CRLEntry` for that certificate by calling `getRevokedCertificate( )`. Call `getThisUpdate( )` to obtain the date this CRL was issued. Use `getNextUpdate( )` to find if the CRL has been superseded by a newer version. Use `getRevokedCertificates( )` to obtain a `Set` of all `X509CRLEntry` objects from this CRL.

Obtain an `X509CRL` object by creating a `CertificateFactory` for certificate type "X.509" and then using the `generateCRL( )` to parse an X.509 CRL from a stream of bytes. Finally, cast the `CRL` returned by this method to an `X509CRL`.

Figure 14-73. java.security.cert.X509CRL


```
public abstract class X509CRL extends CRL implements X509Extension {
// Protected Constructors
     protected X509CRL( );
// Public Instance Methods
     public abstract byte[ ] getEncoded( ) throws CRLException;
     public abstract java.security.Principal getIssuerDN( );
1.4  public javax.security.auth.x500.X500Principal getIssuerX500Principal( );
     public abstract java.util.Date getNextUpdate( );
5.0  public X509CRLEntry getRevokedCertificate(X509Certificate certificate);
     public abstract X509CRLEntry
        getRevokedCertificate(java.math.BigInteger serialNumber);
     public abstract java.util.Set<? extends X509CRLEntry>
        getRevokedCertificates( );
     public abstract String getSigAlgName( );
     public abstract String getSigAlgOID( );
     public abstract byte[ ] getSigAlgParams( );
     public abstract byte[ ] getSignature( );
     public abstract byte[ ] getTBSCertList( ) throws CRLException;
     public abstract java.util.Date getThisUpdate( );
     public abstract int getVersion( );
     public abstract void verify(java.security.PublicKey key)
        throws CRLException, java.security.NoSuchAlgorithmException,
        java.security.InvalidKeyException, java.security.NoSuchProviderException, java.security.SignatureException;
     public abstract void verify(java.security.PublicKey key, String sigProvider)
        throws CRLException,
        java.security.NoSuchAlgorithmException, java.security.InvalidKeyException,
        java.security.NoSuchProviderException, java.security.SignatureException;
// Public Methods Overriding Object
     public boolean equals(Object other);
     public int hashCode( );
}
```

X509CRLEntry java.security.cert

Java 1.2

This class represents a single entry in an `X509CRL`. It contains the serial number and revocation date for a revoked certificate.

Figure 14-74. java.security.cert.X509CRLEntry


```
public abstract class X509CRLEntry implements X509Extension {
// Public Constructors
     public X509CRLEntry( );
// Public Instance Methods
5.0  public javax.security.auth.x500.X500Principal
        getCertificateIssuer( );     constant
     public abstract byte[ ] getEncoded( ) throws CRLException;
     public abstract java.util.Date getRevocationDate( );
     public abstract java.math.BigInteger getSerialNumber( );
     public abstract boolean hasExtensions( );
// Public Methods Overriding Object
     public boolean equals(Object other);
     public int hashCode( );
     public abstract String toString( );
}
```

Returned By

`X509CRL.getRevokedCertificate( )`

X509CRLSelector java.security.cert

Java 1.4 ***cloneable***

This class is a `CRLSelector` implementation for X.509 CRLs. The various `set` methods allow you to specify criteria that the `match( )` method will use to accept or reject `CRL` objects. Use `addIssuerName( )` to specify the distinguished name of an acceptable issuer for the CRL, or use `setIssuerNames( )` or `setIssuers( )` to specify a `Collection` of valid issuers. Use `setDateAndTime( )` to specify a `Date` for which the CRL must be valid. Use `setMinCRLNumber( )` and `setMaxCRLNumber( )` to set bounds on the sequence number of the CRL. If you are selecting a `CRL` in order to check for revocation of a particular `X509Certificate`, pass that certificate to

`setCertificateChecking( )`. This method does not actually constrain the returned `CRL` objects, but it may help a `CertStore` optimize its search for a relevant `CRL`.

Figure 14-75. java.security.cert.X509CRLSelector


```
public class X509CRLSelector implements CRLSelector {
// Public Constructors
     public X509CRLSelector( );
// Public Instance Methods
5.0  public void addIssuer(javax.security.auth.x500.X500Principal issuer);
     public void addIssuerName(String name) throws java.io.IOException;
     public void addIssuerName(byte[ ] name) throws java.io.IOException;
     public X509Certificate getCertificateChecking( ); default:null
     public java.util.Date getDateAndTime( );  default:null
     public java.util.Collection<Object> getIssuerNames( );  default:null
5.0  public java.util.Collection<javax.security.auth.x500.X500Principal>
        getIssuers( );     default:null
     public java.math.BigInteger getMaxCRL( );    default:null
     public java.math.BigInteger getMinCRL( );    default:null
     public void setCertificateChecking(X509Certificate cert);
     public void setDateAndTime(java.util.Date dateAndTime);
     public void setIssuerNames(java.util.Collection<?> names)
        throws java.io.IOException;
5.0  public void setIssuers(java.util.Collection
        <javax.security.auth.x500.X500Principal> issuers);
     public void setMaxCRLNumber(java.math.BigInteger maxCRL);
     public void setMinCRLNumber(java.math.BigInteger minCRL);
// Methods Implementing CRLSelector
     public Object clone( );
     public boolean match(CRL crl);
// Public Methods Overriding Object
     public String toString( );
}
```

X509Extension java.security.cert

Java 1.2

This interface defines methods for handling a set of extensions to X.509 certificates and CRLs. Each extension has a name, or OID (object identifier), that identifies the type of the extension. An extension may be marked critical or noncritical. Noncritical extensions whose OIDs are not recognized can safely be ignored. However, if a critical exception is not recognized, the `Certificate` or `CRL` should be rejected. Each extension in the set has a byte array of data as its value. The interpretation of these bytes depends on the OID of the extension, of course. Specific extensions are defined by the X.509 and related standards and their details are beyond the scope of this reference.

```
public interface X509Extension {
// Public Instance Methods
     java.util.Set<String> getCriticalExtensionOIDs( );
```


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

```
    byte[ ] getExtensionValue(String oid);
    java.util.Set<String> getNonCriticalExtensionOIDs( );
    boolean hasUnsupportedCriticalExtension( );
}
```

Implementations

`X509Certificate`, `X509CRL`, `X509CRLEntry`

Package java.security.interfaces

Java 1.1

As its name implies, the `java.security.interfaces` package contains only interfaces. These interfaces define methods that provide algorithm-specific information (such as key values and initialization parameter values) about DSA, RSA, and EC public and private keys. If you are using the RSA algorithm, for example, and working with a `java.security.PublicKey` object, you can cast that `PublicKey` to an `RSAPublicKey` object and use the RSA-specific methods defined by `RSAPublicKey` to query the key value directly.

The `java.security.interfaces` package was introduced in Java 1.1. As of Java 1.2, the `java.security.spec` package is the preferred way for obtaining algorithm-specific information about keys and algorithm parameters. This package remains useful in Java 1.2 and later, however, for identifying the type of a given `PublicKey` or `PrivateKey` object.

The interfaces in this package are typically of interest only to programmers who are implementing a security provider or who want to implement cryptographic algorithms themselves. Use of this package typically requires some familiarity with the mathematics underlying DSA and RSA public-key cryptography.

Interfaces

```
public interface DSAKey;
public interface DSAKeyPairGenerator;
public interface DSAParams;
public interface DSAPrivateKey extends DSAKey, java.security.PrivateKey;
public interface DSAPublicKey extends DSAKey, java.security.PublicKey;
public interface ECKey;
public interface ECPrivateKey extends ECKey, java.security.PrivateKey;
public interface ECPublicKey extends ECKey, java.security.PublicKey;
public interface RSAKey;
public interface RSAMultiPrimePrivateCrtKey extends RSAPrivateKey;
public interface RSAPrivateCrtKey extends RSAPrivateKey;
public interface RSAPrivateKey extends java.security.PrivateKey, RSAKey;
public interface RSAPublicKey extends java.security.PublicKey, RSAKey;
```


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

DSAKey java.security.interfaces

Java 1.1

This interface defines a method that must be implemented by both public and private DSA keys.

```
public interface DSAKey {
// Public Instance Methods
     DSAParams getParams( );
}
```

Implementations

`DSAPrivateKey`, `DSAPublicKey`

DSAKeyPairGenerator java.security.interfaces

Java 1.1

This interface defines algorithm-specific `KeyPairGenerator` initialization methods for DSA keys. To generate a pair of DSA keys, use the static `getInstance( )` factory method of `java.security.KeyPairGenerator` and specify "DSA" as the desired algorithm name. If you wish to perform DSA-specific initialization, cast the returned `KeyPairGenerator` to a `DSAKeyPairGenerator` and call one of the `initialize( )` methods defined by this interface. Finally, generate the keys by calling `generateKeyPair( )` on the `KeyPairGenerator`.

```
public interface DSAKeyPairGenerator {
// Public Instance Methods
     void initialize(DSAParams params, java.security.SecureRandom random)
        throws java.security.InvalidParameterException;
     void initialize(int modlen, boolean genParams,
        java.security.SecureRandom random)
        throws java.security.InvalidParameterException;
}
```

DSAParams java.security.interfaces

Java 1.1

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

This interface defines methods for obtaining the DSA parameters `g`, `p`, and `q`. These methods are useful only if you wish to perform cryptographic computation yourself. Using these methods requires a detailed understanding of the mathematics underlying DSA public-key cryptography.

```
public interface DSAParams {
// Public Instance Methods
     java.math.BigInteger getG( );
     java.math.BigInteger getP( );
     java.math.BigInteger getQ( );
}
```

Implementations

`java.security.spec.DSAParameterSpec`

Passed To

`DSAKeyPairGenerator.initialize( )`

Returned By

`DSAKey.getParams( )`

DSAPrivateKey java.security.interfaces

Java 1.1 ***serializable***

This interface represents a DSA private key and provides direct access to the underlying key value. If you are working with a private key you know is a DSA key, you can cast the `PrivateKey` to a `DSAPrivateKey`.

Figure 14-76. java.security.interfaces.DSAPrivateKey


```
public interface DSAPrivateKey extends DSAKeyjava.security.PrivateKey {
// Public Constants
1.2  public static final long serialVersionUID;  =7776497482533790279
// Public Instance Methods
     java.math.BigInteger getX( );
}
```

DSAPublicKey java.security.interfaces

Java 1.1 ***serializable***

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

This interface represents a DSA public key and provides direct access to the underlying key value. If you are working with a public key you know is a DSA key, you can cast the `PublicKey` to a `DSAPublicKey`.

Figure 14-77. java.security.interfaces.DSAPublicKey


```
public interface DSAPublicKey extends DSAKeyjava.security.PublicKey {
// Public Constants
1.2  public static final long serialVersionUID;  =1234526332779022332
// Public Instance Methods
     java.math.BigInteger getY( );
}
```

ECKey **java.security.interfaces**

Java 5.0

This interface defines the API that must be implemented by all elliptic curve keys.

```
public interface ECKey {
// Public Instance Methods
     java.security.spec.ECParameterSpec getParams( );
}
```

Implementations

`ECPrivateKey`, `ECPublicKey`

ECPrivateKey **java.security.interfaces**

Java 5.0 ***serializable***

This interface defines an API that must be implemented by all elliptic curve private keys.

Figure 14-78. java.security.interfaces.ECPrivateKey


```
public interface ECPrivateKey extends ECKeyjava.security.PrivateKey {
// Public Constants
     public static final long serialVersionUID;  =-7896394956925609184
```

```
// Public Instance Methods
     java.math.BigInteger getS( );
}
```

ECPublicKey java.security.interfaces

Java 5.0 ***serializable***

This interface defines an API that must be implemented by all elliptic curve public keys.

Figure 14-79. java.security.interfaces.ECPublicKey


```
public interface ECPublicKey extends ECKeyjava.security.PublicKey {
// Public Constants
     public static final long serialVersionUID;  =-3314988629879632826
// Public Instance Methods
     java.security.spec.ECPoint getW( );
}
```

RSAKey java.security.interfaces

Java 1.3

This is a superinterface for `RSAPublicKey` and `RSAPrivateKey`; it defines a method shared by both classes. Prior to Java 1.3, the `getModulus( )` method was defined independently by `RSAPublicKey` and `RSAPrivateKey`.

```
public interface RSAKey {
// Public Instance Methods
     java.math.BigInteger getModulus( );
}
```

Implementations

`RSAPrivateKey`, `RSAPublicKey`

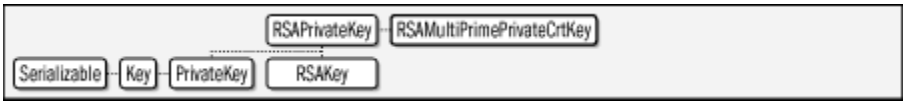
RSAMultiPrimePrivateCrtKey java.security.interfaces

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 interface extends `RSAPrivateKey` and provides a decomposition of the private key into the various numbers used to create it. This interface is very similar to `RSAPrivateCrtKey`, except that it is used to represent RSA private keys that are based on more than two prime factors, and implements the addition `getOtherPrimeInfo( )` method to return information about these additional prime numbers.

Figure 14-80. java.security.interfaces.RSAMultiPrimePrivateCrtKey




```
public interface RSAMultiPrimePrivateCrtKey extends RSAPrivateKey {
// Public Constants
5.0  public static final long serialVersionUID;  =6180585335346280008
// Public Instance Methods
     java.math.BigInteger getCrtCoefficient( );
     java.security.spec.RSAOtherPrimeInfo[ ] getOtherPrimeInfo( );
     java.math.BigInteger getPrimeExponentP( );
     java.math.BigInteger getPrimeExponentQ( );
     java.math.BigInteger getPrimeP( );
     java.math.BigInteger getPrimeQ( );
     java.math.BigInteger getPublicExponent( );
}
```

RSAPrivateCrtKey **java.security.interfaces**

Java 1.2 ***serializable***

This interface extends `RSAPrivateKey` and provides a decomposition (based on the Chinese remainder theorem) of the private-key value into the various pieces that comprise it. This interface is useful only if you plan to implement your own cryptographic algorithms. To use this interface, you must have a detailed understanding of the mathematics underlying RSA public-key cryptography. Given a `java.security.PrivateKey` object, you can use the `instanceof` operator to determine whether you can safely cast it to an `RSAPrivateCrtKey`.

Figure 14-81. java.security.interfaces.RSAPrivateCrtKey

```
public interface RSAPrivateCrtKey extends RSAPrivateKey {
// Public Constants
5.0  public static final long serialVersionUID;  =-5682214253527700368
// Public Instance Methods
     java.math.BigInteger getCrtCoefficient( );
     java.math.BigInteger getPrimeExponentP( );
     java.math.BigInteger getPrimeExponentQ( );
     java.math.BigInteger getPrimeP( );
     java.math.BigInteger getPrimeQ( );
     java.math.BigInteger getPublicExponent( );
}
```

RSAPrivateKey java.security.interfaces

Java 1.2 ***serializable***

This interface represents an RSA private key and provides direct access to the underlying key values. If you are working with a private key you know is an RSA key, you can cast the `PrivateKey` to an `RSAPrivateKey`.

Figure 14-82. java.security.interfaces.RSAPrivateKey


```
public interface RSAPrivateKey extends java.security.PrivateKeyRSAKey {
// Public Constants
5.0  public static final long serialVersionUID;  =5187144804936595022
// Public Instance Methods
     java.math.BigInteger getPrivateExponent( );
}
```

Implementations

`RSAMultiPrimePrivateCrtKey`, `RSAPrivateCrtKey`

RSAPublicKey java.security.interfaces

Java 1.2 ***serializable***

This interface represents an RSA public key and provides direct access to the underlying key values. If you are working with a public key you know is an RSA key, you can cast the `PublicKey` to an `RSAPublicKey`.

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

Figure 14-83. java.security.interfaces.RSAPublicKey


```
public interface RSAPublicKey extends java.security.PublicKeyRSAKey {
// Public Constants
5.0  public static final long serialVersionUID;  =-8727434096241101194
// Public Instance Methods
     java.math.BigInteger getPublicExponent( );
}
```

Package java.security.spec

Java 1.2

The `java.security.spec` package contains classes that define transparent representations for DSA, RSA, and EC public and private keys and for X.509 and PKCS#8 encodings of those keys. It also defines a transparent representation for DSA algorithm parameters. The classes in this package are used in conjunction with `java.security.KeyFactory` and `java.security.AlgorithmParameters` for converting opaque `Key` and `AlgorithmParameters` objects to and from transparent representations.

This package is not frequently used. To make use of it, you must be somewhat familiar with the mathematics that underlies DSA and RSA public-key encryption and the encoding standards that specify how keys are encoded as byte streams.

Interfaces

```
public interface AlgorithmParameterSpec;
public interface ECField;
public interface KeySpec;
```

Classes

```
public class DSAParameterSpec implements AlgorithmParameterSpec,
        java.security.interfaces.DSAParams;
public class DSAPrivateKeySpec implements KeySpec;
public class DSAPublicKeySpec implements KeySpec;
public class ECFieldF2m implements ECField;
public class ECFieldFp implements ECField;
public class ECGenParameterSpec implements AlgorithmParameterSpec;
public class ECParameterSpec implements AlgorithmParameterSpec;
public class ECPoint;
public class ECPrivateKeySpec implements KeySpec;
public class ECPublicKeySpec implements KeySpec;
public class EllipticCurve;
public abstract class EncodedKeySpec implements KeySpec;
   public class PKCS8EncodedKeySpec extends EncodedKeySpec;
   public class X509EncodedKeySpec extends EncodedKeySpec;
public class MGF1ParameterSpec implements AlgorithmParameterSpec;
public class PSSParameterSpec implements AlgorithmParameterSpec;
```


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

```
public class RSAKeyGenParameterSpec implements AlgorithmParameterSpec;
public class RSAOtherPrimeInfo;
public class RSAPrivateKeySpec implements KeySpec;
   public class RSAMultiPrimePrivateCrtKeySpec extends RSAPrivateKeySpec;
   public class RSAPrivateCrtKeySpec extends RSAPrivateKeySpec;
public class RSAPublicKeySpec implements KeySpec;
```

Exceptions

```
public class InvalidKeySpecException
        extends java.security.GeneralSecurityException;
public class InvalidParameterSpecException
        extends java.security.GeneralSecurityException;
```

AlgorithmParameterSpec java.security.spec

Java 1.2

This interface defines no methods; it marks classes that define a transparent representation of cryptographic parameters. You can use an `AlgorithmParameterSpec` object to initialize an opaque `java.security.AlgorithmParameters` object.

```
public interface AlgorithmParameterSpec {
}
```

Implementations

`DSAParameterSpec`, `ECGenParameterSpec`, `ECParameterSpec`, `MGF1ParameterSpec`, `PSSParameterSpec`, `RSAKeyGenParameterSpec`, `javax.crypto.spec.DHGenParameterSpec`, `javax.crypto.spec.DHParameterSpec`, `javax.crypto.spec.IvParameterSpec`, `javax.crypto.spec.OAEPParameterSpec`, `javax.crypto.spec.PBEParameterSpec`, `javax.crypto.spec.RC2ParameterSpec`, `javax.crypto.spec.RC5ParameterSpec`

Passed To

Too many methods to list.

Returned By

`java.security.AlgorithmParameters.getParameterSpec( )`, `java.security.AlgorithmParametersSpi.engineGetParameterSpec( )`, `PSSParameterSpec.getMGFParameters( )`, `javax.crypto.Cipher.getMaxAllowedParameterSpec( )`, `javax.crypto.spec.OAEPParameterSpec.getMGFParameters( )`

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

DSAParameterSpec java.security.spec

Java 1.2

This class represents algorithm parameters used with DSA public-key cryptography.

Figure 14-84. java.security.spec.DSAParameterSpec


```
public class DSAParameterSpec implements AlgorithmParameterSpec,
        java.security.interfaces.DSAParams {
// Public Constructors
     public DSAParameterSpec(java.math.BigInteger p, java.math.BigInteger q,
        java.math.BigInteger g);
// Methods Implementing DSAParams
     public java.math.BigInteger getG( );
     public java.math.BigInteger getP( );
     public java.math.BigInteger getQ( );
}
```

DSAPrivateKeySpec java.security.spec

Java 1.2

This class is a transparent representation of a DSA private key.

Figure 14-85. java.security.spec.DSAPrivateKeySpec


```
public class DSAPrivateKeySpec implements KeySpec {
// Public Constructors
     public DSAPrivateKeySpec(java.math.BigInteger x, java.math.BigInteger p,
        java.math.BigInteger q, java.math.BigInteger g);
// Public Instance Methods
     public java.math.BigInteger getG( );
     public java.math.BigInteger getP( );
     public java.math.BigInteger getQ( );
     public java.math.BigInteger getX( );
}
```

DSAPublicKeySpec java.security.spec

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

Java 1.2

This class is a transparent representation of a DSA public key.

Figure 14-86. java.security.spec.DSAPublicKeySpec


```
public class DSAPublicKeySpec implements KeySpec {
// Public Constructors
     public DSAPublicKeySpec(java.math.BigInteger y, java.math.BigInteger p,
        java.math.BigInteger q, java.math.BigInteger g);
// Public Instance Methods
     public java.math.BigInteger getG( );
     public java.math.BigInteger getP( );
     public java.math.BigInteger getQ( );
     public java.math.BigInteger getY( );
}
```

ECField java.security.spec

Java 5.0

This interface represents a "finite field" for elliptic curve cryptography.

```
public interface ECField {
// Public Instance Methods
     int getFieldSize( );
}
```

Implementations

`ECFieldF2m`, `ECFieldFp`

Passed To

`EllipticCurve.EllipticCurve( )`

Returned By

`EllipticCurve.getField( )`

ECFieldF2m java.security.spec

Java 5.0

This class defines an immutable representation of a "characteristic 2 finite field" for elliptic curve cryptography.

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

Figure 14-87. java.security.spec.ECFieldF2m


```
public class ECFieldF2m implements ECField {
// Public Constructors
     public ECFieldF2m(int m);
     public ECFieldF2m(int m, int[ ] ks);
     public ECFieldF2m(int m, java.math.BigInteger rp);
// Public Instance Methods
     public int getM( );
     public int[ ] getMidTermsOfReductionPolynomial( );
     public java.math.BigInteger getReductionPolynomial( );
// Methods Implementing ECField
     public int getFieldSize( );
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
}
```

ECFieldFp java.security.spec

Java 5.0

This class defines an immutable representation of a "prime finite field" for elliptic curve cryptography.

Figure 14-88. java.security.spec.ECFieldFp


```
public class ECFieldFp implements ECField {
// Public Constructors
     public ECFieldFp(java.math.BigInteger p);
// Public Instance Methods
     public java.math.BigInteger getP( );
// Methods Implementing ECField
     public int getFieldSize( );
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
}
```

ECGenParameterSpec java.security.spec

Java 5.0

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

This class specifies parameters for generating elliptic curve domain parameters.

Figure 14-89. java.security.spec.ECGenParameterSpec


```
public class ECGenParameterSpec implements AlgorithmParameterSpec {
// Public Constructors
     public ECGenParameterSpec(String stdName);
// Public Instance Methods
     public String getName( );
}
```

ECParameterSpec java.security.spec

Java 5.0

This class defines an immutable representation for a set of parameters for elliptic curve cryptography.

Figure 14-90. java.security.spec.ECParameterSpec


```
public class ECParameterSpec implements AlgorithmParameterSpec {
// Public Constructors
     public ECParameterSpec(EllipticCurve curve, ECPoint g,
        java.math.BigInteger n, int h);
// Public Instance Methods
     public int getCofactor( );
     public EllipticCurve getCurve( );
     public ECPoint getGenerator( );
     public java.math.BigInteger getOrder( );
}
```

Passed To

`ECPrivateKeySpec.ECPrivateKeySpec( ), ECPublicKeySpec.ECPublicKeySpec( )`

Returned By

`java.security.interfaces.ECKey.getParams( ), ECPrivateKeySpec.getParams( ),ECPublicKeySpec.getParams( )`

ECPoint java.security.spec

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

Java 5.0

This class defines an immutable representation of a point on an elliptic curve, using affine coordinates.

```
public class ECPoint {
// Public Constructors
     public ECPoint(java.math.BigInteger x, java.math.BigInteger y);
// Public Constants
     public static final ECPoint POINT_INFINITY;
// Public Instance Methods
     public java.math.BigInteger getAffineX( );
     public java.math.BigInteger getAffineY( );
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
}
```

Passed To

`ECParameterSpec.ECParameterSpec( ), ECPublicKeySpec.ECPublicKeySpec( )`

Returned By

`java.security.interfaces.ECPublicKey.getW( ), ECParameterSpec.getGenerator( ),ECPublicKeySpec.getW( )`

ECPrivateKeySpec java.security.spec

Java 5.0

This class is an immutable representation of a private key for elliptic curve cryptography.

Figure 14-91. java.security.spec.ECPrivateKeySpec


```
public class ECPrivateKeySpec implements KeySpec {
// Public Constructors
     public ECPrivateKeySpec(java.math.BigInteger s, ECParameterSpec params);
// Public Instance Methods
     public ECParameterSpec getParams( );
     public java.math.BigInteger getS( );
}
```

ECPublicKeySpec java.security.spec

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

Java 5.0

This class is an immutable representation of a public key for elliptic curve cryptography.

Figure 14-92. java.security.spec.ECPublicKeySpec


```
public class ECPublicKeySpec implements KeySpec {
// Public Constructors
     public ECPublicKeySpec(ECPoint w, ECParameterSpec params);
// Public Instance Methods
     public ECParameterSpec getParams( );
     public ECPoint getW( );
}
```

EllipticCurve java.security.spec

Java 5.0

This class is an immutable representation of an elliptic curve. See `ECParameterSpec`.

```
public class EllipticCurve {
// Public Constructors
     public EllipticCurve(ECField field, java.math.BigInteger a,
        java.math.BigInteger b);
     public EllipticCurve(ECField field, java.math.BigInteger a,
        java.math.BigInteger b, byte[ ] seed);
// Public Instance Methods
     public java.math.BigInteger getA( );
     public java.math.BigInteger getB( );
     public ECField getField( );
     public byte[ ] getSeed( );
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
}
```

Passed To

`ECParameterSpec.ECParameterSpec( )`

Returned By

`ECParameterSpec.getCurve( )`

EncodedKeySpec java.security.spec

Java 1.2

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

This abstract class represents a public or private key in an encoded format. It serves as the superclass for encoding-specific classes.

Figure 14-93. java.security.spec.EncodedKeySpec


```
public abstract class EncodedKeySpec implements KeySpec {
// Public Constructors
     public EncodedKeySpec(byte[ ] encodedKey);
// Public Instance Methods
     public byte[ ] getEncoded( );
     public abstract String getFormat( );
}
```

Subclasses

`PKCS8EncodedKeySpec`, `X509EncodedKeySpec`

InvalidKeySpecException — java.security.spec

Java 1.2 — ***serializable checked***

Signals a problem with a `KeySpec`.

Figure 14-94. java.security.spec.InvalidKeySpecException


```
public class InvalidKeySpecException
       extends java.security.GeneralSecurityException {
// Public Constructors
     public InvalidKeySpecException( );
5.0  public InvalidKeySpecException(Throwable cause);
     public InvalidKeySpecException(String msg);
5.0  public InvalidKeySpecException(String message, Throwable cause);
}
```

Thrown By

`java.security.KeyFactory.{generatePrivate( ),generatePublic( ), getKeySpec( )},java.security.KeyFactorySpi. {engineGeneratePrivate( ),engineGeneratePublic( ), engineGetKeySpec( )}, javax.crypto.EncryptedPrivateKeyInfo.getKeySpec( ), javax.crypto.SecretKeyFactory.{generateSecret( ),getKeySpec( )},`

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

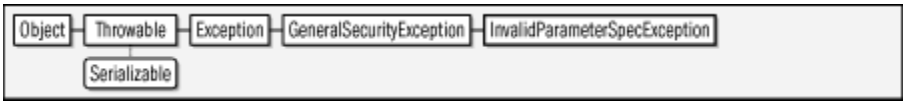
`javax.crypto.SecretKeyFactorySpi.{engineGenerateSecret( ), engineGetKeySpec( )}`

InvalidParameterSpecException java.security.spec

Java 1.2 ***serializable checked***

Signals a problem with an `AlgorithmParameterSpec`.

Figure 14-95. java.security.spec.InvalidParameterSpecException




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

Thrown By

`java.security.AlgorithmParameters.{getParameterSpec( ),init( )}, java.security.AlgorithmParametersSpi.{engineGetParameterSpec( ), engineInit( )}`

KeySpec java.security.spec

Java 1.2

This interface defines no methods; it marks classes that define a transparent representation of a cryptographic key. Use a `java.security.KeyFactory` to convert a `KeySpec` to and from an opaque `java.security.Key`.

```
public interface KeySpec {
}
```

Implementations

`DSAPrivateKeySpec`, `DSAPublicKeySpec`, `ECPrivateKeySpec`, `ECPublicKeySpec`, `EncodedKeySpec`, `RSAPrivateKeySpec`, `RSAPublicKeySpec`, `javax.crypto.spec.DESedeKeySpec`, `javax.crypto.spec.DESKeySpec`, `javax.crypto.spec.DHPrivateKeySpec`,

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

javax.crypto.spec.DHPublicKeySpec, javax.crypto.spec.PBEKeySpec, javax.crypto.spec.SecretKeySpec

Passed To

java.security.KeyFactory.{generatePrivate( ), generatePublic( )}, java.security.KeyFactorySpi.{engineGeneratePrivate( ), engineGeneratePublic( )}, javax.crypto.SecretKeyFactory.generateSecret( ), javax.crypto.SecretKeyFactorySpi.engineGenerateSecret( )

Returned By

java.security.KeyFactory.getKeySpec( ), java.security.KeyFactorySpi.engineGetKeySpec( ), javax.crypto.SecretKeyFactory.getKeySpec( ), javax.crypto.SecretKeyFactorySpi.engineGetKeySpec( )

MGF1ParameterSpec — java.security.spec

Java 5.0

This class represents parameters for "mask generation function" MGF1 of the OAEP Padding and RSA-PSS signature scheme, defined in the PKCS #1 standard, version 2.1. The constants represent predefined instances of the class, whose digest algorithm matches the constant name.

Figure 14-96. java.security.spec.MGF1ParameterSpec


```
public class MGF1ParameterSpec implements AlgorithmParameterSpec {
// Public Constructors
     public MGF1ParameterSpec(String mdName);
// Public Constants
     public static final MGF1ParameterSpec SHA1;
     public static final MGF1ParameterSpec SHA256;
     public static final MGF1ParameterSpec SHA384;
     public static final MGF1ParameterSpec SHA512;
// Public Instance Methods
     public String getDigestAlgorithm( );
}
```

PKCS8EncodedKeySpec — java.security.spec

Java 1.2

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

This class represents a private key, encoded according to the PKCS#8 standard.

Figure 14-97. java.security.spec.PKCS8EncodedKeySpec


```
public class PKCS8EncodedKeySpec extends EncodedKeySpec {
// Public Constructors
     public PKCS8EncodedKeySpec(byte[ ] encodedKey);
// Public Methods Overriding EncodedKeySpec
     public byte[ ] getEncoded( );
     public final String getFormat( );
}
```

Returned By

`javax.crypto.EncryptedPrivateKeyInfo.getKeySpec( )`

PSSParameterSpec java.security.spec

Java 1.4

This class represents algorithm parameters used with the RSA PSS encoding scheme, which is defined by version 2.1 of the RSA standard PKCS#1. This class has been substantially enhanced in Java 5.0.

Figure 14-98. java.security.spec.PSSParameterSpec


```
public class PSSParameterSpec implements AlgorithmParameterSpec {
// Public Constructors
     public PSSParameterSpec(int saltLen);
5.0  public PSSParameterSpec(String mdName, String mgfName,
        AlgorithmParameterSpec mgfSpec,
        int saltLen, int trailerField);
// Public Constants
5.0  public static final PSSParameterSpec DEFAULT;
// Public Instance Methods
5.0  public String getDigestAlgorithm( );
5.0  public String getMGFAlgorithm( );
5.0  public AlgorithmParameterSpec getMGFParameters( );
     public int getSaltLength( );
5.0  public int getTrailerField( );
}
```

RSAKeyGenParameterSpec java.security.spec

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

Java 1.3

This class represents parameters that generate public/private key pairs for RSA cryptography.

Figure 14-99. java.security.spec.RSAKeyGenParameterSpec


```
public class RSAKeyGenParameterSpec implements AlgorithmParameterSpec {
// Public Constructors
     public RSAKeyGenParameterSpec(int keysize,
        java.math.BigInteger publicExponent);
// Public Constants
     public static final java.math.BigInteger F0;
     public static final java.math.BigInteger F4;
// Public Instance Methods
     public int getKeysize( );
     public java.math.BigInteger getPublicExponent( );
}
```

RSAMultiPrimePrivateCrtKeySpec java.security.spec

Java 1.4

This class is a transparent representation of a multi-prime RSA private key. It is very similar to `RSAPrivateCrtKeySpec`, but adds an additional method for obtaining information about the other primes associated with the key.

Figure 14-100. java.security.spec.RSAMultiPrimePrivateCrtKeySpec


```
public class RSAMultiPrimePrivateCrtKeySpec extends RSAPrivateKeySpec {
// Public Constructors
     public RSAMultiPrimePrivateCrtKeySpec(java.math.BigInteger modulus,
       java.math.BigInteger publicExponent,
       java.math.BigInteger privateExponent,
       java.math.BigInteger primeP,
       java.math.BigInteger primeQ,
       java.math.BigInteger primeExponentP,
       java.math.BigInteger primeExponentQ,
       java.math.BigInteger crtCoefficient,
       RSAOtherPrimeInfo[ ] otherPrimeInfo);
// Public Instance Methods
     public java.math.BigInteger getCrtCoefficient( );
     public RSAOtherPrimeInfo[ ] getOtherPrimeInfo( );
     public java.math.BigInteger getPrimeExponentP( );
     public java.math.BigInteger getPrimeExponentQ( );
     public java.math.BigInteger getPrimeP( );
```


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

```
    public java.math.BigInteger getPrimeQ( );
    public java.math.BigInteger getPublicExponent( );
}
```

RSAOtherPrimeInfo java.security.spec

Java 1.4

This class represents the (prime, exponent, coefficient) triplet that constitues an "OtherPrimeInfo" structure that is used with RSA multi-prime private keys, as defined in version 2.1 of the PKCS#1 standard.

```
public class RSAOtherPrimeInfo {
// Public Constructors
    public RSAOtherPrimeInfo(java.math.BigInteger prime,
       java.math.BigInteger primeExponent,
       java.math.BigInteger crtCoefficient);
// Public Instance Methods
    public final java.math.BigInteger getCrtCoefficient( );
    public final java.math.BigInteger getExponent( );
    public final java.math.BigInteger getPrime( );
}
```

Passed To

`RSAMultiPrimePrivateCrtKeySpec.RSAMultiPrimePrivateCrtKeySpec( )`

Returned By

`java.security.interfaces.RSAMultiPrimePrivateCrtKey.getOtherPrimeInfo( ),RSAMultiPrimePrivateCrtKeySpec.getOtherPrimeInfo( )`

RSAPrivateCrtKeySpec java.security.spec

Java 1.2

This class is a transparent representation of an RSA private key including, for convenience, the Chinese remainder theorem values associated with the key.

Figure 14-101. java.security.spec.RSAPrivateCrtKeySpec


```
public class RSAPrivateCrtKeySpec extends RSAPrivateKeySpec {
// Public Constructors
    public RSAPrivateCrtKeySpec(java.math.BigInteger modulus,
       java.math.BigInteger publicExponent,
       java.math.BigInteger privateExponent,
```


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

```
        java.math.BigInteger primeP,
        java.math.BigInteger primeQ,
        java.math.BigInteger primeExponentP,
        java.math.BigInteger primeExponentQ,
        java.math.BigInteger crtCoefficient);
// Public Instance Methods
     public java.math.BigInteger getCrtCoefficient( );
     public java.math.BigInteger getPrimeExponentP( );
     public java.math.BigInteger getPrimeExponentQ( );
     public java.math.BigInteger getPrimeP( );
     public java.math.BigInteger getPrimeQ( );
     public java.math.BigInteger getPublicExponent( );
}
```

RSAPrivateKeySpec java.security.spec

Java 1.2

This class is a transparent representation of an RSA private key.

Figure 14-102. java.security.spec.RSAPrivateKeySpec


```
public class RSAPrivateKeySpec implements KeySpec {
// Public Constructors
     public RSAPrivateKeySpec(java.math.BigInteger modulus,
        java.math.BigInteger privateExponent);
// Public Instance Methods
     public java.math.BigInteger getModulus( );
     public java.math.BigInteger getPrivateExponent( );
}
```

Subclasses

`RSAMultiPrimePrivateCrtKeySpec`, `RSAPrivateCrtKeySpec`

RSAPublicKeySpec java.security.spec

Java 1.2

This class is a transparent representation of an RSA public key.

Figure 14-103. java.security.spec.RSAPublicKeySpec


```
public class RSAPublicKeySpec implements KeySpec {
// Public Constructors
```

```
     public RSAPublicKeySpec(java.math.BigInteger modulus,
        java.math.BigInteger publicExponent);
// Public Instance Methods
     public java.math.BigInteger getModulus( );
     public java.math.BigInteger getPublicExponent( );
}
```

X509EncodedKeySpec java.security.spec

Java 1.2

This class represents a public or private key encoded according to the X.509 standard.

Figure 14-104. java.security.spec.X509EncodedKeySpec


```
public class X509EncodedKeySpec extends EncodedKeySpec {
// Public Constructors
     public X509EncodedKeySpec(byte[ ] encodedKey);
// Public Methods Overriding EncodedKeySpec
     public byte[ ] getEncoded( );
     public final String getFormat( );
}
```