



CHAPTER 15

java.security and Subpackages

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

`java.security`

This large package 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.acl`

This package implements an infrastructure for Access Control Lists. It is not used by the Java security mechanism and has been superseded by classes in `java.security`.

`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 generic 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.

Package `java.security`

Java 1.1

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

Package java.security

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 pseudo-random 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 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 sub-packages. 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 Principal;  
public interface PrivateKey extends Key;  
public interface PrivilegedAction;  
public interface PrivilegedExceptionAction;  
public interface PublicKey extends Key;
```

Collections:

public abstract class **Provider** extends java.util.Properties;

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 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 KeyStore;  
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;  
    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 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;
```

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;
```

Package *java.security*

```
public class KeyStoreException extends GeneralSecurityException;
public class NoSuchAlgorithmException extends GeneralSecurityException;
public class NoSuchProviderException extends GeneralSecurityException;
public class SignatureException extends GeneralSecurityException;
public class UnrecoverableKeyException extends GeneralSecurityException;
public class InvalidParameterException extends IllegalArgumentException;
public class PrivilegedActionException extends Exception;
public class ProviderException extends RuntimeException;
```

AccessControlContext

Java 1.2

java.security

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

Passed To: `AccessControlContext.AccessControlContext()`, `AccessController.doPrivileged()`, `javax.security.auth.Subject.doAsPrivileged()`, `getSubject()`

Returned By: `AccessController.getContext()`

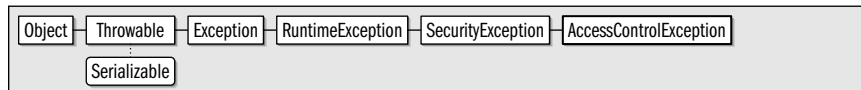
AccessControlException

Java 1.2

java.security

serializable unchecked

This is 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.



```
public class AccessControlException extends SecurityException {
    // Public Constructors
    public AccessControlException(String s);
    public AccessControlException(String s, java.security.Permission p);
}
```

```
// Public Instance Methods
public java.security.Permission getPermission();
}
```

Thrown By: AccessControlContext.checkPermission(), AccessController.checkPermission()

AccessController

Java 1.2

java.security

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(java.security.Permission perm) throws AccessControlException;
public static Object doPrivileged(PrivilegedExceptionAction action) throws PrivilegedActionException;   native
public static Object doPrivileged(PrivilegedAction action);   native
public static Object doPrivileged(PrivilegedExceptionAction action, AccessControlContext context)   native
    throws PrivilegedActionException;
public static Object doPrivileged(PrivilegedAction action, AccessControlContext context);   native
public static AccessControlContext getContext();
}
```

AlgorithmParameterGenerator

Java 1.2

java.security

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.

AlgorithmParameterGenerator

```
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);  
}
```

Returned By: AlgorithmParameterGenerator.getInstance()

AlgorithmParameterGeneratorSpi

Java 1.2

java.security

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

Passed To: AlgorithmParameterGenerator.AlgorithmParameterGenerator()

AlgorithmParameters

Java 1.2

java.security

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 java.security.spec.AlgorithmParameterSpec getParameterSpec(Class 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();
}

```

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(), AlgorithmParameters.getInstance(),
 Signature.getParameters(), SignatureSpi.engineGetParameters(), javax.crypto.Cipher.getParameters(),
 javax.crypto.CipherSpi.engineGetParameters(), javax.crypto.EncryptedPrivateKeyInfo.getAlgParameters()

AlgorithmParametersSpi

Java 1.2

java.security

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 java.security.spec.AlgorithmParameterSpec engineGetParameterSpec(Class 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 1.2

java.security

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.



```

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

```

BasicPermission

Java 1.2

java.security

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.



```

public abstract class BasicPermission extends java.security.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(java.security.Permission p);
    public PermissionCollection newPermissionCollection();
}

```

Subclasses: `java.awt.AWTPermission`, `java.io.SerializablePermission`, `RuntimePermission`, `java.lang.reflect.ReflectPermission`, `java.net.NetPermission`, `SecurityPermission`, `java.sql.SQLPermission`, `java.util.PropertyPermission`, `java.util.logging.LoggingPermission`, `javax.net.ssl.SSLPermission`, `javax.security.auth.AuthPermission`, `javax.security.auth.kerberos.DelegationPermission`, `javax.sound.sampled.AudioPermission`

Certificate**Java 1.1; Deprecated in Java 1.2****java.security**

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 also `java.security.cert.Certificate`.

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

Passed To: `Identity.{addCertificate(), removeCertificate()}`

Returned By: `Identity.certificates()`

CodeSource**Java 1.2****java.security****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. 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.



```
public class CodeSource implements Serializable {
    // Public Constructors
    public CodeSource(java.net.URL url, java.security.cert.Certificate[] certs);
    // Public Instance Methods
    public final java.security.cert.Certificate[] getCertificates();
    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();
}
```

Passed To: `java.net.URLClassLoader.getPermissions(), CodeSource.implies(), java.security.Policy.getPermissions(), ProtectionDomain.ProtectionDomain(), SecureClassLoader.{defineClass(), getPermissions()}, javax.security.auth.Policy.getPermissions()`

Returned By: `ProtectionDomain.getCodeSource()`

DigestException

DigestException

Java 1.1

java.security

serializable checked

This exception signals a problem creating a message digest.



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

Thrown By: MessageDigest.digest(), MessageDigestSpi.engineDigest()

DigestInputStream

Java 1.1

java.security

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`.



```
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 1.1

java.security

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`.



```

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

DomainCombiner

Java 1.3

java.security

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
    public abstract ProtectionDomain[] combine(ProtectionDomain[] currentDomains,
                                                ProtectionDomain[] assignedDomains);
}
  
```

Implementations: `javax.security.auth.SubjectDomainCombiner`

Passed To: `AccessControlContext.AccessControlContext()`

Returned By: `AccessControlContext.getDomainCombiner()`

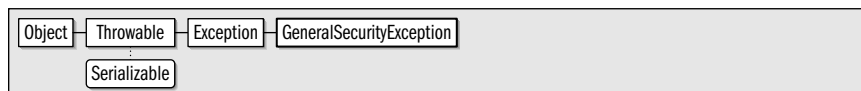
GeneralSecurityException

Java 1.2

java.security

serializable checked

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



```

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

GeneralSecurityException

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

Subclasses: Too many classes to list.

Guard

Java 1.2

java.security

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  
    public abstract void checkGuard(Object object) throws SecurityException;  
}
```

Implementations: `java.security.Permission`

Passed To: `GuardedObject.GuardedObject()`

GuardedObject

Java 1.2

java.security

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 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.

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

Identity

Java 1.1; Deprecated in Java 1.2

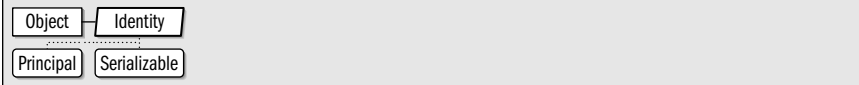
java.security

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 such as 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.



```

classDiagram
    class Identity {
        <<abstract>>
        +Principal
        +Serializable
        +getInfo() String
        +setInfo(String info) void
        +getName() String
        +getPublicKey() PublicKey
        +setPublicKey(PublicKey key) void
        +getScope() IdentityScope
        +addCertificate(Certificate certificate) void
        +certificates() Certificate[]
        +removeCertificate(Certificate certificate) void
        +toString(boolean detailed) String
        +equals(Object identity) boolean
        +hashCode() int
        +identityEquals(Identity identity) boolean
    }
    Identity <|-- IdentityScope
    Identity <|-- Signer
  
```

```

public abstract class Identity implements java.security.Principal, Serializable {
// Public Constructors
    public Identity(String name);
    public Identity(String name, IdentityScope scope) throws KeyManagementException;
// Protected Constructors
    protected Identity();
// Property Accessor Methods (by property name)
    public String getInfo();
    public void setInfo(String info);
    public final String getName(); // Implements:Principal
    public PublicKey getPublicKey();
    public void setPublicKey(PublicKey key) throws KeyManagementException;
    public final IdentityScope getScope();
// Public Instance Methods
    public void addCertificate(java.security.Certificate certificate) throws KeyManagementException;
    public java.security.Certificate[ ] certificates();
    public void removeCertificate(java.security.Certificate certificate) 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: Identity.identityEquals(), IdentityScope.{addIdentity(), removeIdentity()}

Returned By: IdentityScope.getIdentity()


IdentityScope

Java 1.1; Deprecated in Java 1.2

java.security

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.



```

classDiagram
    class IdentityScope {
        <<abstract>>
        +Identity
        +Principal
        +Serializable
        +IdentityScope(String name)
        +IdentityScope(String name, IdentityScope scope)
    }
    IdentityScope <|-- IdentityScope
  
```

```

public abstract class IdentityScope extends Identity {
// Public Constructors
    public IdentityScope(String name);
    public IdentityScope(String name, IdentityScope scope) throws KeyManagementException;
}
  
```

IdentityScope

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

Passed To: Identity.Identity(), IdentityScope.{IdentityScope(), setSystemScope()}, Signer.Signer()

Returned By: Identity.getScope(), IdentityScope.getSystemScope()

InvalidAlgorithmParameterException

Java 1.2

java.security

serializable checked

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



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

Thrown By: Too many methods to list.

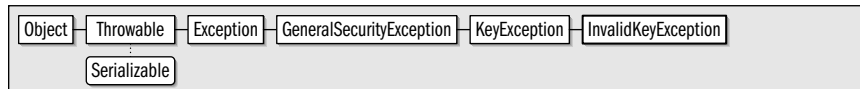
InvalidKeyException

Java 1.1

java.security

serializable checked

This exception signals that a Key is not valid.



```
public class InvalidKeyException extends KeyException {
// Public Constructors
    public InvalidKeyException();
    public InvalidKeyException(String msg);
}
```

Thrown By: Too many methods to list.

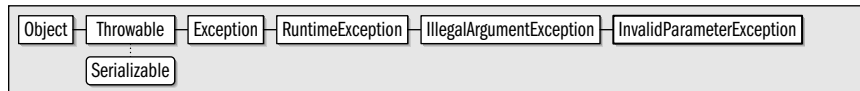
InvalidParameterException

Java 1.1

java.security

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.



```

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()

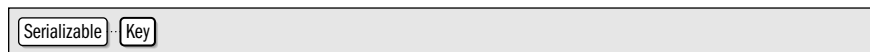
Key

Java 1.1

java.security

serializable

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().



```

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

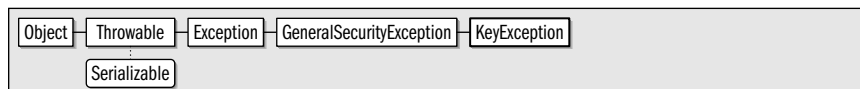
KeyException

Java 1.1

java.security

serializable checked

This exception signals that something is wrong with a key. See also the subclasses InvalidKeyException and KeyManagementException.



```

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

KeyException

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

Subclasses: InvalidKeyException, KeyManagementException

Thrown By: java.security.Certificate.{decode(), encode()}, Signer.setKeyPair()

KeyFactory

Java 1.2

java.security

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;  
    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 java.security.spec.KeySpec getKeySpec(Key key, Class keySpec)  
        throws java.security.spec.InvalidKeySpecException;  
    public final Provider getProvider();  
    public final Key translateKey(Key key) throws InvalidKeyException;  
}
```

Returned By: `KeyFactory.getInstance()`

KeyFactorySpi

Java 1.2

java.security

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 java.security.spec.KeySpec engineGetKeySpec(Key key, Class keySpec)
        throws java.security.spec.InvalidKeySpecException;
    protected abstract Key engineTranslateKey(Key key) throws InvalidKeyException;
}

```

Passed To: KeyFactory.KeyFactory()

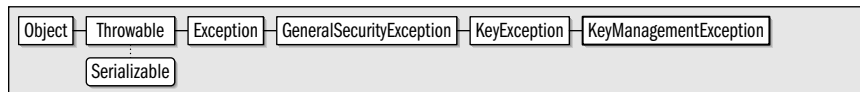
KeyManagementException

Java 1.1

java.security

serializable checked

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



```

public class KeyManagementException extends KeyException {
// Public Constructors
    public KeyManagementException();
    public KeyManagementException(String msg);
}

```

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 1.1

java.security

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.



```

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()

KeyPairGenerator

Java 1.1

java.security

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.

Object	KeyPairGeneratorSpi	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) empty
        throws InvalidAlgorithmParameterException;
    public void initialize(int keysize, SecureRandom random); empty
}

```

Returned By: `KeyPairGenerator.getInstance()`

KeyPairGeneratorSpi

Java 1.2

java.security

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

KeyStore

Java 1.2

java.security

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.

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 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.

```

public class KeyStore {
    // Protected Constructors
    protected KeyStore(KeyStoreSpi keyStoreSpi, Provider provider, String type);
    // 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 aliases() throws KeyStoreException;
    public final boolean containsAlias(String alias) throws KeyStoreException;
    public final void deleteEntry(String alias) throws KeyStoreException;
}

```

KeyStore

```
public final java.security.cert.Certificate getCertificate(String alias) 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;
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;
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;
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;
public final void store(java.io.OutputStream stream, char[] password) throws KeyStoreException,
    java.io.IOException, NoSuchAlgorithmException, java.security.cert.CertificateException;
}
```

Passed To: 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.getInstance()

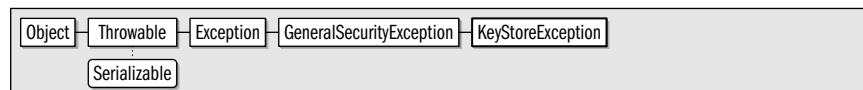
KeyStoreException

Java 1.2

java.security

serializable checked

This exception signals a problem with a KeyStore.



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

Thrown By: Too many methods to list.

KeyStoreSpi

Java 1.2

java.security

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 engineAliases();
public abstract boolean engineContainsAlias(String alias);
public abstract void engineDeleteEntry(String alias) throws KeyStoreException;
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);
public abstract Key engineGetKey(String alias, char[] password) throws NoSuchAlgorithmException,
    UnrecoverableKeyException;
public abstract boolean engineIsCertificateEntry(String alias);
public abstract boolean engineIsKeyEntry(String alias);
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;
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();
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 1.1

java.security

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. Finally, 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.

```

Object — MessageDigestSpi — MessageDigest
public abstract class MessageDigest extends MessageDigestSpi {
// Protected Constructors
    protected MessageDigest(String algorithm);
// Public Class Methods
    public static MessageDigest getInstance(String algorithm) throws NoSuchAlgorithmException;
}

```

MessageDigest

```
1.4 public static MessageDigest getInstance(String algorithm, Provider provider) throws NoSuchAlgorithmException;
public static MessageDigest getInstance(String algorithm, String provider) throws NoSuchAlgorithmException,
    NoSuchProviderException;
    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);
    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(),
MessageDigest.getInstance()

Type Of: DigestInputStream.digest, DigestOutputStream.digest

MessageDigestSpi

Java 1.2

java.security

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
    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();
    protected abstract void engineReset();
    protected abstract void engineUpdate(byte input);
    protected abstract void engineUpdate(byte[] input, int offset, int len);
}
```

Subclasses: MessageDigest

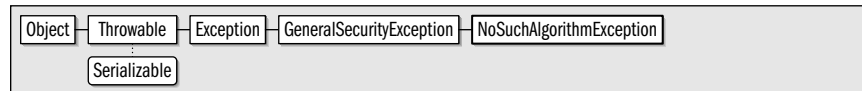
NoSuchAlgorithmException

Java 1.1

java.security

serializable checked

This signals that a requested cryptographic algorithm is not available. It is thrown by `getInstance()` factory methods throughout the `java.security` package.



```

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

Thrown By: Too many methods to list.

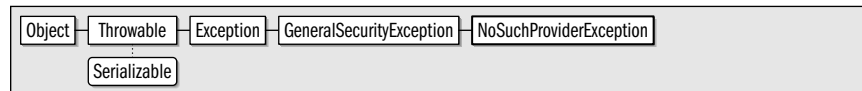
NoSuchProviderException

Java 1.1

java.security

serializable checked

This signals that a requested cryptographic service provider is not available. It is thrown by `getInstance()` factory methods throughout the `java.security` package.



```

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

Thrown By: Too many methods to list.

Permission

Java 1.2

java.security

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 sub-classes 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. With the introduction 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.

Permission

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 `"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.



```
classDiagram
    class Permission {
        +String name
        +String[] actions
        +boolean implies(Permission permission)
        +PermissionCollection newPermissionCollection()
        +void checkGuard(Object object) throws SecurityException
        +boolean equals(Object obj)
        +int hashCode()
        +String toString()
    }
    class Guard
    class Serializable
    Permission <|-- Guard
    Permission <|-- Serializable
```

```
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(java.security.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 1.2

`java.security`

serializable

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.



```
classDiagram
    class PermissionCollection {
        +Permission[] permissions
    }
    class Object
    class Serializable
    PermissionCollection <|-- Object
    PermissionCollection <|-- Serializable
```

```
public abstract class PermissionCollection implements Serializable {
    // Public Constructors
    public PermissionCollection();
}
```



```
// Public Instance Methods
public abstract void add(java.security.Permission permission);
public abstract java.util Enumeration elements();
public abstract boolean implies(java.security.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 1.2

java.security

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.



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

Policy

Java 1.2

java.security

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.

Policy

In 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`.

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

Passed To: `java.security.Policy.setPolicy()`

Returned By: `java.security.Policy.getPolicy()`

Principal

Java 1.1

`java.security`

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

Implementations: `Identity`, `java.security.acl.Group`, `javax.security.auth.kerberos.KerberosPrincipal`, `javax.security.auth.x500.X500Principal`

Passed To: Too many methods to list.

Returned By: `java.security.Certificate.{getGuarantor(), getPrincipal()}`,
`ProtectionDomain.getPrincipals()`, `java.security.acl.AclEntry.getPrincipal()`,
`java.security.cert.X509Certificate.{getIssuerDN(), getSubjectDN()}`,
`java.security.cert.X509CRL.getIssuerDN()`, `javax.security.cert.X509Certificate.{getIssuerDN(), getSubjectDN()}`

PrivateKey

Java 1.1

`java.security`

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`.

Serializable Key PrivateKey

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

Implementations: java.security.interfaces.DSAPrivateKey, java.security.interfaces.RSAPrivateKey, javax.crypto.interfaces.DHPrivateKey

Passed To: KeyPair.KeyPair(), Signature.initSign(), SignatureSpi.engineInitSign(), SignedObject.SignedObject(), javax.security.auth.x500.X500PrivateKeyCredential.X500PrivateKeyCredential()

Returned By: KeyFactory.generatePrivate(), KeyFactorySpi.engineGeneratePrivate(), KeyPair.getPrivate(), Signer.getPrivateKey(), javax.net.ssl.X509KeyManager.getPrivateKey(), javax.security.auth.x500.X500PrivateKeyCredential.getPrivateKey()

PrivilegedAction

Java 1.2

java.security

This interface defines a block of code (the run() method) that is to be executed as privileged code by the AccessController.doPrivileged() method. When privileged code is run in this way, 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 {
    // Public Instance Methods
    public abstract Object run();
}
```

Passed To: AccessController.doPrivileged(), javax.security.auth.Subject.{doAs(), doAsPrivileged()}

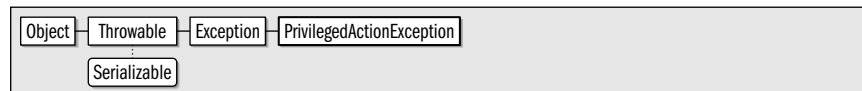
PrivilegedActionException

Java 1.2

java.security

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



```
public class PrivilegedActionException extends Exception {
    // Public Constructors
    public PrivilegedActionException(Exception exception);
}
```

PrivilegedActionException

```
// 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

Java 1.2

java.security

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

```
public interface PrivilegedExceptionAction {
// Public Instance Methods
public abstract Object run() throws Exception;
}
```

Passed To: AccessController.doPrivileged(), javax.security.auth.Subject.{doAs(), doAsPrivileged()}

ProtectionDomain

Java 1.2

java.security

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 permissions 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 (to add new permissions for specific users, for example) 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,
java.security.Principal[] principals);
// Public Instance Methods
1.4 public final ClassLoader getClassLoader();
public final CodeSource getCodeSource();
public final PermissionCollection getPermissions();
1.4 public final java.security.Principal[] getPrincipals();
public boolean implies(java.security.Permission permission);
}
```

```
// Public Methods Overriding Object
public String toString();
}
```

Passed To: `ClassLoader.defineClass()`, `AccessControlContext.AccessControlContext()`, `DomainCombiner.combine()`, `java.security.Policy.{getPermissions(), implies()}`, `javax.security.auth.SubjectDomainCombiner.combine()`

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

Provider

Java 1.1

java.security

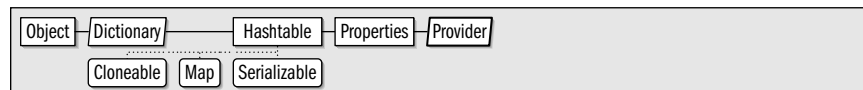
cloneable serializable collection

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”.

Security providers are installed for a Java system 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 need to install custom providers.



```
public abstract class Provider extends java.util.Properties {
```

```
// Protected Constructors
```

```
protected Provider(String name, double version, String info);
```

```
// Public Instance Methods
```

```
public String getInfo();
```

```
public String getName();
```

```
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
```

Provider

```
1.2 public java.util.Set entrySet(); synchronized
1.2 public java.util.Set 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 values();
}
```

Passed To: Too many methods to list.

Returned By: Too many methods to list.

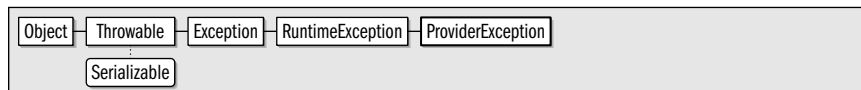
ProviderException

Java 1.1

java.security

serializable unchecked

This 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.



```
public class ProviderException extends RuntimeException {
// Public Constructors
    public ProviderException();
    public ProviderException(String s);
}
```

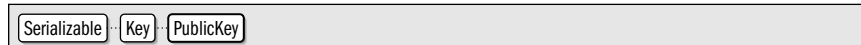
PublicKey

Java 1.1

java.security

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`.



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

Implementations: `java.security.interfaces.DSAPublicKey`, `java.security.interfaces.RSAPublicKey`, `javax.crypto.interfaces.DHPPublicKey`

Passed To: Too many methods to list.

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()`, `javax.security.cert.Certificate.getPublicKey()`

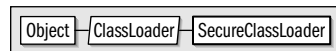
SecureClassLoader

Java 1.2

java.security

This class adds two protected methods to those defined by `ClassLoader`. The `defineClass()` method is passed a `CodeSource` object that represents the source of the class being loaded. It calls the `getPermissions()` method to obtain a `PermissionCollection` for that `CodeSource`. It 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.



```

public class SecureClassLoader extends ClassLoader {
    // Protected Constructors
    protected SecureClassLoader();
    protected SecureClassLoader(ClassLoader parent);
    // Protected Instance Methods
    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 1.1

java.security

serializable

This class generates cryptographic-quality pseudo-random 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 pseudo-random 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 pseudo-random bytes. You can also call any of the methods defined by the `Random` 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 pseudo-random 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 pseudo-random generators. `generateSeed()` may use a different algorithm than `nextBytes()` and may produce higher-quality randomness, usually at the expense of increased computation time.



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);  
    1.2 public final Provider getProvider(); default:Sun  
    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.

Returned By: SecureRandom.getInstance()

Type Of: SignatureSpi.appRandom

SecureRandomSpi

Java 1.2

java.security

serializable

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



```
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 1.1

java.security

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 that return only providers that implement the algorithm or algorithms specified in a `String` or `Map` object exist. 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 class name 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 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 getAlgorithms(String serviceName);
    public static String getProperty(String key);
    public static Provider getProvider(String name);                                synchronized
    public static Provider[] getProviders();                                       synchronized
    1.3 public static Provider[] getProviders(java.util.Map 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);
}
```

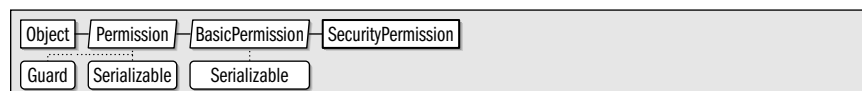
SecurityPermission

Java 1.2

`java.security`

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.



SecurityPermission

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

Signature

Java 1.1

java.security

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 bytes to be signed or verified. 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.



```
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;  
    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;  
    public final void update(byte[] data) throws SignatureException;
```

```

    public final void update(byte b) 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()}

Returned By: Signature.getInstance()

SignatureException

Java 1.1

java.security

serializable checked

This signals a problem while creating or verifying a digital signature.



```

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

```

Thrown By: Too many methods to list.

SignatureSpi

Java 1.2

java.security

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;
}

```

SignatureSpi

```
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

SignedObject

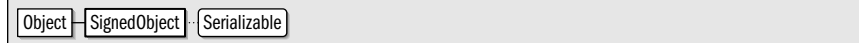
Java 1.2

java.security

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.



```
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 1.1; Deprecated in Java 1.2

java.security

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



```
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();
}

```

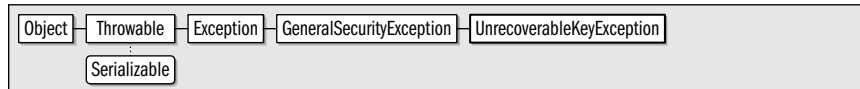
UnrecoverableKeyException

Java 1.2

java.security

serializable checked

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



```

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

```

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

UnresolvedPermission

Java 1.2

java.security

serializable permission

This class is used internally to provide a mechanism for delayed resolution of permissions. An `UnresolvedPermission` holds a textual representation of a `Permission` object that can later be used to create the actual `Permission` object. Applications never need to use this class.



```

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

```

Package *java.security.acl*

Java 1.1

The *java.security.acl* package defines, but does not implement, an incomplete framework for working with access control lists (ACLs). This package was added in Java 1.1, but has been superseded in Java 1.2 by the access-control mechanisms of the *java.security* package. In particular, see the *Permission* and *Policy* classes of that package. The use of this package is not recommended.

Interfaces:

```
public interface Acl extends Owner;
public interface AclEntry extends Cloneable;
public interface Group extends java.security.Principal;
public interface Owner;
public interface Permission;
```

Exceptions:

```
public class AclNotFoundException extends Exception;
public class LastOwnerException extends Exception;
public class NotOwnerException extends Exception;
```

Acl

Java 1.1

java.security.acl

This interface represents an *access control list*, or ACL. An ACL is a list of *AclEntry* objects; most of the methods of this class manage that list. The exception is the *checkPermission()* method that tests whether this ACL grants a specified *java.security.acl.Permission* to a specified *java.security.Principal*. Note that *Acl* extends *Owner*. The methods of the *Owner* interface maintain a list of ACL owners. Only owners are allowed to modify an ACL.



```
public interface Acl extends Owner {
    // Public Instance Methods
    public abstract boolean addEntry(java.security.Principal caller, AclEntry entry) throws NotOwnerException;
    public abstract boolean checkPermission(java.security.Principal principal,
                                             java.security.acl.Permission permission);
    public abstract java.util.Enumeration entries();
    public abstract String getName();
    public abstract java.util.Enumeration getPermissions(java.security.Principal user);
    public abstract boolean removeEntry(java.security.Principal caller, AclEntry entry) throws NotOwnerException;
    public abstract void setName(java.security.Principal caller, String name) throws NotOwnerException;
    public abstract String toString();
}
```

AclEntry

Java 1.1

java.security.acl

cloneable

This interface defines a single entry of an ACL. Each *AclEntry* represents a set of *java.security.acl.Permission* objects either granted or denied to a given *java.security.Principal*. By

default, an `AclEntry` represents permissions granted to the principal. Call `setNegativePermissions()` if you want the `AclEntry` to represent a set of permissions to be denied.

Cloneable AclEntry

```
public interface AclEntry extends Cloneable {
// Public Instance Methods
    public abstract boolean addPermission(java.security.acl.Permission permission);
    public abstract boolean checkPermission(java.security.acl.Permission permission);
    public abstract Object clone();
    public abstract java.security.Principal getPrincipal();
    public abstract boolean isNegative();
    public abstract java.util.Enumeration permissions();
    public abstract boolean removePermission(java.security.acl.Permission permission);
    public abstract void setNegativePermissions();
    public abstract boolean setPrincipal(java.security.Principal user);
    public abstract String toString();
}
```

Passed To: `Acl`.{`addEntry()`, `removeEntry()`}

AclNotFoundException

Java 1.1

java.security.acl

serializable checked

This signals that the specified `Acl` could not be found. Note that none of the interfaces in `java.security.acl` throw this exception; it is provided for the benefit of `Acl` implementations.

Object Throwable Exception AclNotFoundException
Serializable

```
public class AclNotFoundException extends Exception {
// Public Constructors
    public AclNotFoundException();
}
```

Group

Java 1.1

java.security.acl

This interface represents a set, or group, of `java.security.Principal` objects. The methods of the interface serve to manage the membership of the group. Note that `Group` extends the `Principal` interface, and, therefore, you can use a `Group` object wherever you would use a `Principal` object in this package.

Principal Group

```
public interface Group extends java.security.Principal {
// Public Instance Methods
    public abstract boolean addMember(java.security.Principal user);
    public abstract boolean isMember(java.security.Principal member);
    public abstract java.util.Enumeration members();
    public abstract boolean removeMember(java.security.Principal user);
}
```

LastOwnerException

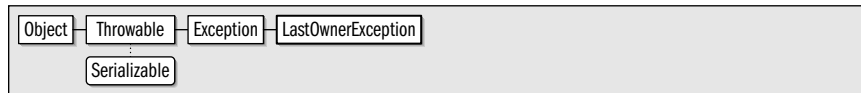
LastOwnerException

Java 1.1

java.security.acl

serializable checked

This signals that an Acl or Owner has only one Principal remaining in its ownership list and that this single owner cannot be removed.



```
public class LastOwnerException extends Exception {  
    // Public Constructors  
    public LastOwnerException();  
}
```

Thrown By: Owner.deleteOwner()

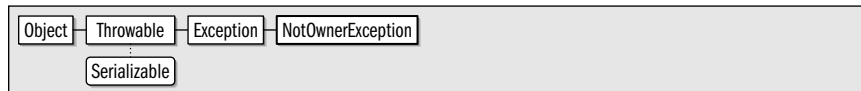
NotOwnerException

Java 1.1

java.security.acl

serializable checked

This exception is thrown by various methods of Acl and Owner when they are called by a Principal that is not an owner.



```
public class NotOwnerException extends Exception {  
    // Public Constructors  
    public NotOwnerException();  
}
```

Thrown By: Acl.{addEntry(), removeEntry(), setName()}, Owner.{addOwner(), deleteOwner()}

Owner

Java 1.1

java.security.acl

This interface represents the owner or owners of an ACL. The interface defines methods for managing and checking membership in the list of owners.

```
public interface Owner {  
    // Public Instance Methods  
    public abstract boolean addOwner(java.security.Principal caller, java.security.Principal owner)  
        throws NotOwnerException;  
    public abstract boolean deleteOwner(java.security.Principal caller, java.security.Principal owner)  
        throws NotOwnerException, LastOwnerException;  
    public abstract boolean isOwner(java.security.Principal owner);  
}
```

Implementations: Acl

Permission

Java 1.1

java.security.acl

This interface represents a permission. The meaning of the permission is entirely up to the implementation. Do not confuse this interface with the newer java.security.Permission class. Also note that this interface does not have the implies() method of java.security.Permission and is therefore significantly less versatile.


```

public interface Permission {
    // Public Instance Methods
    public abstract boolean equals(Object another);
    public abstract String toString();
}

```

Passed To: Acl.checkPermission(), AclEntry.{addPermission(), checkPermission(), removePermission()}

Package **java.security.cert**

Java 1.2

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 as well as `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 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;

```

Package java.security.cert

```
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 final class PolicyQualifierInfo;
public class TrustAnchor;
public class X509CertSelector implements CertSelector;
public abstract class X509CRLEntry implements X509Extension;
public class X509CRLSelector implements CRLSelector;
```

Protected Inner Classes:

```
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 1.2

java.security.cert

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 defined in Java 1.1 and deprecated in Java 1.2.

```

Object Certificate Serializable
public abstract class Certificate implements Serializable {
// Protected Constructors
    protected Certificate(String type);
// Inner Classes
1.3 protected static class CertificateRep implements Serializable;
// Public Instance Methods
    public abstract byte[] getEncoded() throws java.security.cert.CertificateEncodingException;
    public abstract java.security.PublicKey getPublicKey();
    public final String getType();
    public abstract void verify(java.security.PublicKey key) throws java.security.cert.CertificateException,
        java.security.NoSuchAlgorithmException, java.security.InvalidKeyException,
        java.security.NoSuchProviderException, java.security.SignatureException;
    public abstract void verify(java.security.PublicKey key, String sigProvider)
        throws java.security.cert.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: `java.security.cert.X509Certificate`

Passed To: Too many methods to list.

Returned By: `java.net.JarURLConnection.getCertificates()`, `java.security.CodeSource.getCertificates()`,
`java.security.KeyStore.{getCertificate(), getCertificateChain()}`,
`java.security.KeyStoreSpi.{engineGetCertificate(), engineGetCertificateChain()}`,
`CertificateFactory.generateCertificate()`, `CertificateFactorySpi.engineGenerateCertificate()`,
`java.util.jar.JarEntry.getCertificates()`, `javax.net.ssl.HandshakeCompletedEvent.{getLocalCertificates(),`
`getPeerCertificates()}`, `javax.net.ssl.HttpURLConnection.{getLocalCertificates(), getServerCertificates()}`,
`javax.net.ssl.SSLSession.{getLocalCertificates(), getPeerCertificates()}`

Certificate.CertificateRep

Java 1.3

`java.security.cert`

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

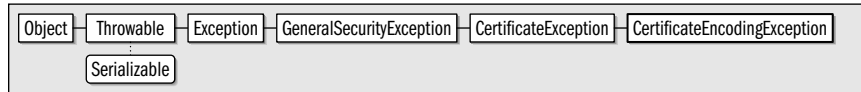
CertificateEncodingException

Java 1.2

java.security.cert

serializable checked

This signals an error while attempting to encode a certificate.



```
public class CertificateEncodingException extends java.security.cert.CertificateException {  
    // Public Constructors  
    public CertificateEncodingException();  
    public CertificateEncodingException(String message);  
}
```

Thrown By: java.security.cert.Certificate.getEncoded(), CertPath.getEncoded(),
java.security.cert.X509Certificate.getTBSCertificate()

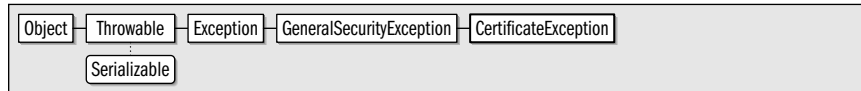
CertificateException

Java 1.2

java.security.cert

serializable checked

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



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

Subclasses: java.security.cert.CertificateEncodingException,
java.security.cert.CertificateExpiredException, java.security.cert.CertificateNotYetValidException,
java.security.cert.CertificateParsingException

Thrown By: Too many methods to list.

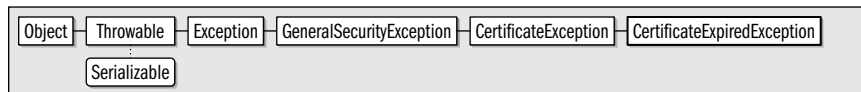
CertificateExpiredException

Java 1.2

java.security.cert

serializable checked

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



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

Thrown By: java.security.cert.X509Certificate.checkValidity()

CertificateFactory

Java 1.2

java.security.cert

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 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** objects 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 “PkiPath” encodings. **getCertPathEncodings()** 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 java.security.cert.CertificateException;
    1.4 public static final CertificateFactory getInstance(String type, java.security.Provider provider)
        throws java.security.cert.CertificateException;
    public static final CertificateFactory getInstance(String type, String provider)
        throws java.security.cert.CertificateException, java.security.NoSuchProviderException;
    // Public Instance Methods
    public final java.security.cert.Certificate generateCertificate(java.io.InputStream inStream)
        throws java.security.cert.CertificateException;
    public final java.util.Collection generateCertificates(java.io.InputStream inStream)
        throws java.security.cert.CertificateException;
    1.4 public final CertPath generateCertPath(java.util.List certificates) throws java.security.cert.CertificateException;
    1.4 public final CertPath generateCertPath(java.io.InputStream inStream)
        throws java.security.cert.CertificateException;
    1.4 public final CertPath generateCertPath(java.io.InputStream inStream, String encoding)
        throws java.security.cert.CertificateException;
```

CertificateFactory

```
public final CRL generateCRL(java.io.InputStream inStream) throws CRLException;  
public final java.util.Collection generateCRLs(java.io.InputStream inStream) throws CRLException;  
1.4 public final java.util.Iterator getCertPathEncodings();  
public final java.security.Provider getProvider();  
public final String getType();  
}
```

Returned By: CertificateFactory.getInstance()

CertificateFactorySpi

Java 1.2

java.security.cert

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 java.security.cert.CertificateException;  
    public abstract java.util.Collection engineGenerateCertificates(java.io.InputStream inStream)  
        throws java.security.cert.CertificateException;  
    1.4 public CertPath engineGenerateCertPath(java.util.List certificates) throws java.security.cert.CertificateException;  
    1.4 public CertPath engineGenerateCertPath(java.io.InputStream inStream)  
        throws java.security.cert.CertificateException;  
    1.4 public CertPath engineGenerateCertPath(java.io.InputStream inStream, String encoding)  
        throws java.security.cert.CertificateException;  
    public abstract CRL engineGenerateCRL(java.io.InputStream inStream) throws CRLException;  
    public abstract java.util.Collection engineGenerateCRLs(java.io.InputStream inStream) throws CRLException;  
    1.4 public java.util.Iterator engineGetCertPathEncodings();  
}
```

Passed To: CertificateFactory.CertificateFactory()

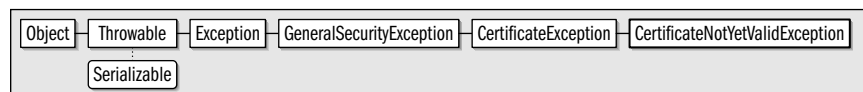
CertificateNotYetValidException

Java 1.2

java.security.cert

serializable checked

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



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

Thrown By: java.security.cert.X509Certificate.checkValidity()

CertificateParsingException

Java 1.2

java.security.cert

serializable checked

This signals an error or other problem while parsing a certificate.



```

public class CertificateParsingException extends java.security.cert.CertificateException {
// Public Constructors
    public CertificateParsingException();
    public CertificateParsingException(String message);
}
  
```

Thrown By: java.security.cert.X509Certificate.{getExtendedKeyUsage(), getIssuerAlternativeNames(), getSubjectAlternativeNames()}

CertPath

Java 1.4

java.security.cert

serializable

A **CertPath** is an 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.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 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 thread-safe.



```

public abstract class CertPath implements Serializable {
// Protected Constructors
    protected CertPath(String type);
// Inner Classes
    protected static class CertPathRep implements Serializable;
}
  
```

CertPath

```
// Public Instance Methods
public abstract java.util.List getCertificates();
public abstract byte[] getEncoded() throws java.security.cert.CertificateEncodingException;
public abstract byte[] getEncoded(String encoding) throws java.security.cert.CertificateEncodingException;
public abstract java.util.Iterator 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;
}
```

Passed To: CertPathValidator.validate(), CertPathValidatorException.CertPathValidatorException(), CertPathValidatorSpi.engineValidate(), PKIXCertPathBuilderResult.PKIXCertPathBuilderResult()

Returned By: CertificateFactory.generateCertPath(), CertificateFactorySpi.engineGenerateCertPath(), CertPathBuilderResult.getCertPath(), CertPathValidatorException.getCertPath(), PKIXCertPathBuilderResult.getCertPath()

CertPath.CertPathRep

Java 1.4

java.security.cert

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 1.4

java.security.cert

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 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 interface 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();
}
```

Returned By: CertPathBuilder.getInstance()

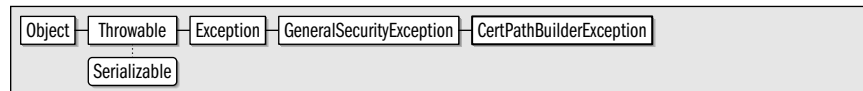
CertPathBuilderException

Java 1.4

java.security.cert

serializable checked

This signals a problem while building a certification path with CertPathBuilder.



```
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);
    // Public Methods Overriding Throwable
    public Throwable getCause(); default:null
    public String getMessage(); default:null
    public void printStackTrace();
    public void printStackTrace(java.io.PrintWriter pw);
    public void printStackTrace(java.io.PrintStream ps);
    public String toString();
}
```

Thrown By: CertPathBuilder.build(), CertPathBuilderSpi.engineBuild()

CertPathBuilderResult

Java 1.4

java.security.cert

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.



```
public interface CertPathBuilderResult extends Cloneable {
    // Public Instance Methods
}
```

CertPathBuilderResult

```
public abstract Object clone();  
public abstract CertPath getCertPath();  
}
```

Implementations: PKIXCertPathBuilderResult

Returned By: CertPathBuilder.build(), CertPathBuilderSpi.engineBuild()

CertPathBuilderSpi

Java 1.4

java.security.cert

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 1.4

java.security.cert

cloneable

CertPathParameters 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.

Cloneable CertPathParameters

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

Implementations: PKIXParameters

Passed To: CertPathBuilder.build(), CertPathBuilderSpi.engineBuild(), CertPathValidator.validate(), CertPathValidatorSpi.engineValidate()

CertPathValidator

Java 1.4

java.security.cert

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

Returned By: `CertPathValidator.getInstance()`

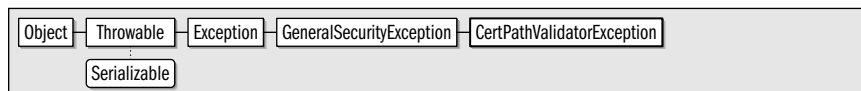
CertPathValidatorException

Java 1.4

`java.security.cert`

serializable checked

This 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).



```
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
    // Public Methods Overriding Throwable
    public Throwable getCause(); default: null
    public String getMessage(); default: null
    public void printStackTrace();
    public void printStackTrace(java.io.PrintStream ps);
    public void printStackTrace(java.io.PrintWriter pw);
}
```

CertPathValidatorException

```
public String toString();  
}
```

Thrown By: CertPathValidator.validate(), CertPathValidatorSpi.engineValidate(),
PKIXCertPathChecker.{check(), init()}

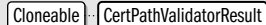
CertPathValidatorResult

Java 1.4

java.security.cert

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 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`.

 `Cloneable` `CertPathValidatorResult`

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

Implementations: PKIXCertPathValidatorResult

Returned By: CertPathValidator.validate(), CertPathValidatorSpi.engineValidate()

CertPathValidatorSpi

Java 1.4

java.security.cert

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 1.4

java.security.cert

cloneable

This interface defines an API for determining whether a `Certificate` meets some criteria. Implementations are used to specify criteria 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`.

 `Cloneable` `CertSelector`

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

```
public abstract 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 1.4

java.security.cert

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 example) of untrusted certificates.

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 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 be used by the **CertPathBuilder** and **CertPathValidator** classes.

All public methods of **CertStore** are thread-safe.

```
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 getCertificates(CertSelector selector) throws CertStoreException;
    public final CertStoreParameters getCertStoreParameters();
    public final java.util.Collection getCRLs(CRLSelector selector) throws CertStoreException;
    public final java.security.Provider getProvider();
    public final String getType();
}
```

Passed To: PKIXParameters.addCertStore()

CertStore

Returned By: CertStore.getInstance()

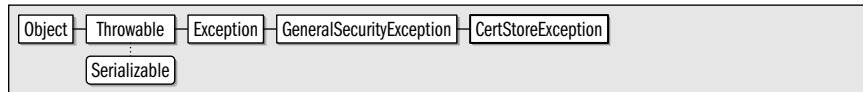
CertStoreException

Java 1.4

java.security.cert

serializable checked

This signals a problem while querying a CertStore for certificates or CRLs.



```
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);
// Public Methods Overriding Throwable
    public Throwable getCause();
    public String getMessage();
    public void printStackTrace();
    public void printStackTrace(java.io.PrintWriter pw);
    public void printStackTrace(java.io.PrintStream ps);
    public String toString();
}
```

Thrown By: CertStore.{getCertificates(), getCRLs()}, CertStoreSpi.{engineGetCertificates(), engineGetCRLs()}

CertStoreParameters

Java 1.4

java.security.cert

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".



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

Implementations: CollectionCertStoreParameters, LDAPCertStoreParameters

Passed To: CertStore.{CertStore(), getInstance()}, CertStoreSpi.CertStoreSpi()

Returned By: CertStore.getCertStoreParameters()

CertStoreSpi

Java 1.4

java.security.cert

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 engineGetCertificates(CertSelector selector) throws CertStoreException;
    public abstract java.util.Collection engineGetCRLs(CRLSelector selector) throws CertStoreException;
}

```

Passed To: CertStore.CertStore()

CollectionCertStoreParameters

Java 1.4

java.security.cert

cloneable

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



```

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();
}

```

CRL

Java 1.2

java.security.cert

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 1.2

java.security.cert

serializable checked

This signals an error or other problem while working with a CRL.



```

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

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

CRLSelector

Java 1.4

java.security.cert

cloneable

This interface defines an API for determining whether a CRL object meets some criteria. Implementations are used to specify criteria 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.



```

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

Implementations: X509CRLSelector

Passed To: CertStore.getCRLs(), CertStoreSpi.engineGetCRLs()

LDAPCertStoreParameters

Java 1.4

java.security.cert

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.



```

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: [quot ] localhost [quot ]
// Methods Implementing CertStoreParameters
public Object clone();
// Public Methods Overriding Object
public String toString();
}

```

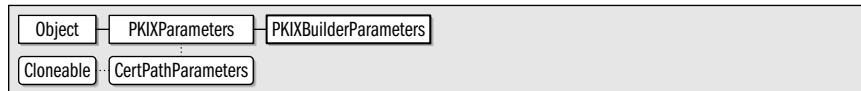
PKIXBuilderParameters

Java 1.4

java.security.cert

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.



```

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 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();
}

```

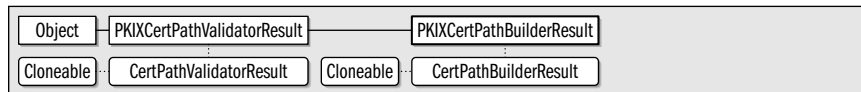
PKIXCertPathBuilderResult

Java 1.4

java.security.cert

cloneable

An instance of this class is returned 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.



```

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();
}

```

PKIXCertPathBuilderResult

```
// Public Methods Overriding PKIXCertPathValidatorResult
public String toString();
}
```


PKIXCertPathChecker

Java 1.4

java.security.cert

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.



```
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 unresolvedCritExts)
    throws CertPathValidatorException;
public abstract java.util.Set 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 1.4

java.security.cert

cloneable

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.



```
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 1.4

java.security.cert

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



```

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 trustAnchors) throws java.security.InvalidAlgorithmParameterException;
// Property Accessor Methods (by property name)
    public boolean isAnyPolicyInhibited();
    public void setAnyPolicyInhibited(boolean val);
    public java.util.List getCertPathCheckers();
    public void setCertPathCheckers(java.util.List checkers);
    public java.util.List getCertStores();
    public void setCertStores(java.util.List stores);
    public java.util.Date getDate();
    public void setDate(java.util.Date date);
    public boolean isExplicitPolicyRequired();
    public void setExplicitPolicyRequired(boolean val);
    public java.util.Set getInitialPolicies();
    public void setInitialPolicies(java.util.Set initialPolicies);
    public boolean isPolicyMappingInhibited();
    public void setPolicyMappingInhibited(boolean val);
    public boolean getPolicyQualifiersRejected();
    public void setPolicyQualifiersRejected(boolean qualifiersRejected);
    public boolean isRevocationEnabled();
    public void setRevocationEnabled(boolean val);
    public String getSigProvider();
}

```

PKIXParameters

```
public void setSigProvider(String sigProvider);
public CertSelector getTargetCertConstraints();
public void setTargetCertConstraints(CertSelector selector);
public java.util.Set getTrustAnchors();
public void setTrustAnchors(java.util.Set trustAnchors) throws java.security.InvalidAlgorithmParameterException;
// Public Instance Methods
public void addCertPathChecker(PKIXCertPathChecker checker);
public void addCertStore(CertStore store);
// Methods Implementing CertPathParameters
public Object clone();
// Public Methods Overriding Object
public String toString();
}
```

Subclasses: PKIXBuilderParameters

PolicyNode

Java 1.4

java.security.cert

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 {
// Property Accessor Methods (by property name)
public abstract java.util.Iterator getChildren();
public abstract boolean isCritical();
public abstract int getDepth();
public abstract java.util.Set getExpectedPolicies();
public abstract PolicyNode getParent();
public abstract java.util.Set getPolicyQualifiers();
public abstract String getValidPolicy();
}
```

Passed To: PKIXCertPathBuilderResult.PKIXCertPathBuilderResult(),
PKIXCertPathValidatorResult.PKIXCertPathValidatorResult()

Returned By: PKIXCertPathValidatorResult.getPolicyTree(), PolicyNode.getParent()

PolicyQualifierInfo

Java 1.4

java.security.cert

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

TrustAnchor

Java 1.4

java.security.cert

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 public key of the CA. You may specify these values explicitly or by passing an **X509Certificate** to the **TrustAnchor()** constructor. Both 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(java.security.cert.X509Certificate trustedCert, byte[] nameConstraints);
    public TrustAnchor(String caName, java.security.PublicKey pubKey, byte[] nameConstraints);
// Public Instance Methods
    public final String getCAName();
    public final java.security.PublicKey getCAPublicKey();
    public final byte[] getNameConstraints();
    public final java.security.cert.X509Certificate getTrustedCert();
// Public Methods Overriding Object
    public String toString();
}
```

Passed To: PKIXCertPathBuilderResult.PKIXCertPathBuilderResult(),
PKIXCertPathValidatorResult.PKIXCertPathValidatorResult()

Returned By: PKIXCertPathValidatorResult.getTrustAnchor()

X509Certificate

Java 1.2

java.security.cert

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 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**.



```
public abstract class X509Certificate extends java.security.cert.Certificate implements X509Extension {
// Protected Constructors
    protected X509Certificate();
// Property Accessor Methods (by property name)
    public abstract int getBasicConstraints();
}
```

X509Certificate

```
1.4 public java.util.List getExtendedKeyUsage() throws java.security.cert.CertificateParsingException;
1.4 public java.util.Collection getIssuerAlternativeNames() throws java.security.cert.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 getSubjectAlternativeNames() throws java.security.cert.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 java.security.cert.CertificateEncodingException;
    public abstract int getVersion();
// Public Instance Methods
    public abstract void checkValidity() throws java.security.cert.CertificateExpiredException,
        java.security.cert.CertificateNotYetValidException;
    public abstract void checkValidity(java.util.Date date) throws java.security.cert.CertificateExpiredException,
        java.security.cert.CertificateNotYetValidException;
}
```

Passed To: TrustAnchor.TrustAnchor(), X509CertSelector.setCertificate(), X509CRLSelector.setCertificateChecking(), javax.net.ssl.X509TrustManager.{checkClientTrusted(), checkServerTrusted()}, javax.security.auth.x500.X500PrivateCredential.X500PrivateCredential()

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 1.4

java.security.cert

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 return `true` only 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()`, or `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()`.



```

public class X509CertSelector implements CertSelector {
    // Public Constructors
    public X509CertSelector();
    // Property Accessor Methods (by property name)
    public byte[] getAuthorityKeyIdentifier(); default:null
    public void setAuthorityKeyIdentifier(byte[] authorityKeyID);
    public int getBasicConstraints(); default:-1
    public void setBasicConstraints(int minMaxPathLen);
    public java.security.cert.X509Certificate getCertificate(); default:null
    public void setCertificate(java.security.cert.X509Certificate cert);
    public java.util.Date getCertificateValid(); default:null
    public void setCertificateValid(java.util.Date certValid);
    public java.util.Set getExtendedKeyUsage(); default:null
    public void setExtendedKeyUsage(java.util.Set keyPurposeSet) throws java.io.IOException;
    public byte[] getIssuerAsBytes() throws java.io.IOException; default:null
    public String getIssuerAsString(); default:null
    public boolean[] getKeyUsage(); default:null
    public void setKeyUsage(boolean[] keyUsage);
    public boolean getMatchAllSubjectAltNames(); default:true
    public void setMatchAllSubjectAltNames(boolean matchAllNames);
    public byte[] getNameConstraints(); default:null
    public void setNameConstraints(byte[] bytes) throws java.io.IOException;
    public java.util.Collection getPathToNames(); default:null
    public void setPathToNames(java.util.Collection names) throws java.io.IOException;
    public java.util.Set getPolicy(); default:null
    public void setPolicy(java.util.Set certPolicySet) throws java.io.IOException;
    public java.util.Date getPrivateKeyValid(); default:null
    public void setPrivateKeyValid(java.util.Date privateKeyValid);
    public java.math.BigInteger getSerialNumber(); default:null
    public void setSerialNumber(java.math.BigInteger serial);
    public java.util.Collection getSubjectAlternativeNames(); default:null
    public void setSubjectAlternativeNames(java.util.Collection names) throws java.io.IOException;
    public byte[] getSubjectAsBytes() throws java.io.IOException; default:null
    public String getSubjectAsString(); default:null
    public byte[] getSubjectKeyIdentifier(); default:null
    public void setSubjectKeyIdentifier(byte[] subjectKeyID);
    public java.security.PublicKey getSubjectPublicKey(); default:null
    public void setSubjectPublicKey(java.security.PublicKey key);
    public void setSubjectPublicKey(byte[] key) throws java.io.IOException;
    public String getSubjectPublicKeyAlgID(); default:null
    public void setSubjectPublicKeyAlgID(String oid) throws java.io.IOException;
    // 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, String name) throws java.io.IOException;
    public void addSubjectAlternativeName(int type, byte[] name) throws java.io.IOException;
    public void setIssuer(String issuerDN) throws java.io.IOException;
    public void setIssuer(byte[] issuerDN) throws java.io.IOException;
    public void setSubject(byte[] subjectDN) throws java.io.IOException;
    public void setSubject(String subjectDN) 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 1.2

java.security.cert

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 when 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.



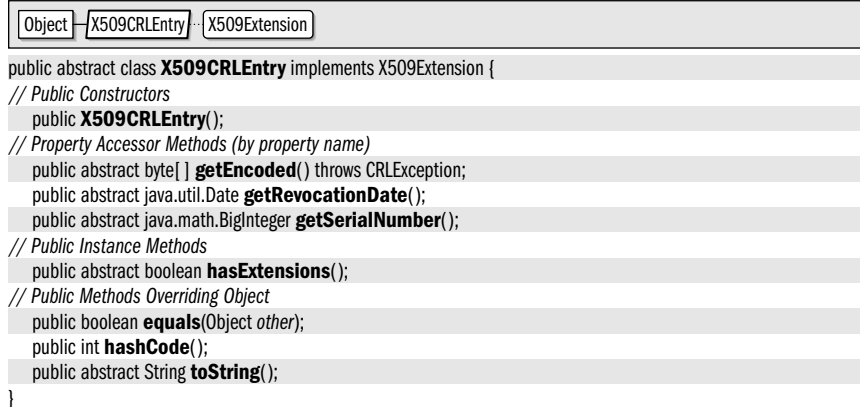
```
public abstract class X509CRL extends CRL implements X509Extension {
// Protected Constructors
protected X509CRL();
// Property Accessor Methods (by property name)
public abstract byte[] getEncoded() throws CRLException;
public abstract java.security.Principal getIssuerDN();
1.4 public abstract javax.security.auth.x500.X500Principal getIssuerX500Principal();
public abstract java.util.Date getNextUpdate();
public abstract java.util.Set 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 Instance Methods
public abstract X509CRLentry getRevokedCertificate(java.math.BigInteger serialNumber);
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 1.2

java.security.cert

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



Returned By: X509CRL.getRevokedCertificate()

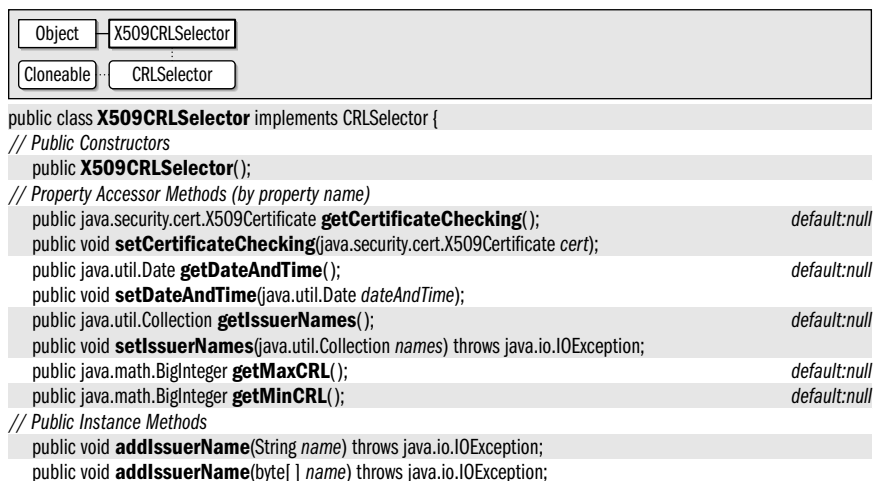
X509CRLSelector

Java 1.4

java.security.cert

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()** to specify a **Collection** of valid issuer names. 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 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.



```

    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 1.2****java.security.cert**

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
    public abstract java.util.Set getCriticalExtensionOIDs();
    public abstract byte[] getExtensionValue(String oid);
    public abstract java.util.Set getNonCriticalExtensionOIDs();
    public abstract boolean hasUnsupportedCriticalExtension();
}

```

Implementations: java.security.cert.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 and RSA 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. In 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, 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. Typically, using this package 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 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;

```

DSAKey

Java 1.1

java.security.interfaces

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

```

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

```

Implementations: DSAPrivateKey, DSAPublicKey

DSAKeyPairGenerator

Java 1.1

java.security.interfaces

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
    public abstract void initialize(DSAParams params, java.security.SecureRandom random)
        throws java.security.InvalidParameterException;
    public abstract void initialize(int modlen, boolean genParams, java.security.SecureRandom random)
        throws java.security.InvalidParameterException;
}

```

DSAParams

Java 1.1

java.security.interfaces

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
    public abstract java.math.BigInteger getG();
    public abstract java.math.BigInteger getP();
    public abstract java.math.BigInteger getQ();
}

```

Implementations: java.security.spec.DSAParameterSpec

DSAParams

Passed To: DSAKeyPairGenerator.initialize()

Returned By: DSAKey.getParams()

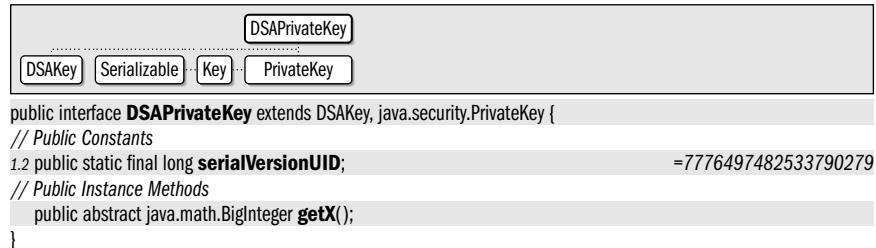
DSAPrivateKey

Java 1.1

java.security.interfaces

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.



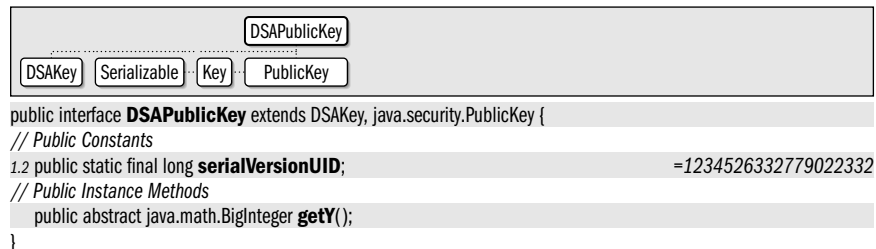
DSAPublicKey

Java 1.1

java.security.interfaces

serializable

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.

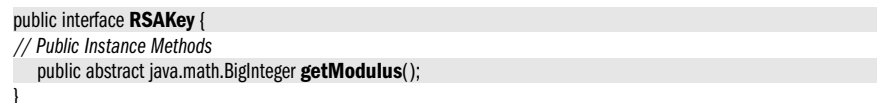


RSAKey

Java 1.3

java.security.interfaces

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.



Implementations: RSAPrivateKey, RSAPublicKey

RSAMultiPrimePrivateCrtKey

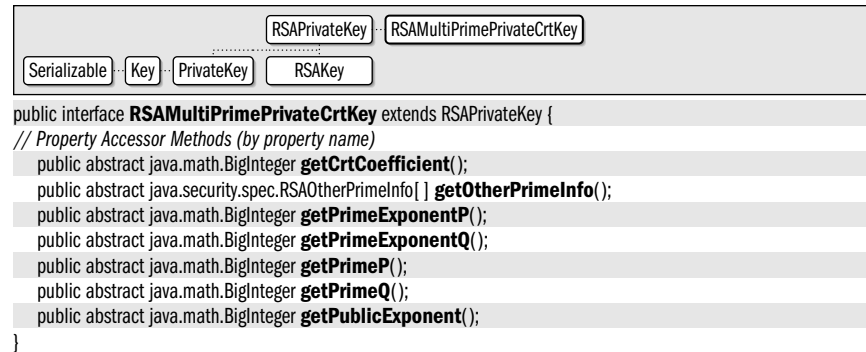
Java 1.4

java.security.interfaces

serializable

This interface extends RSAPrivateKey and provides a decomposition of the private key into the various numbers used to create it. This interface is similar to RSAPrivateCrtKey,

except that it is used to represent RSA private keys that are based on more than two prime factors, and it implements the additional `getOtherPrimeInfo()` method to return information about these additional prime numbers.



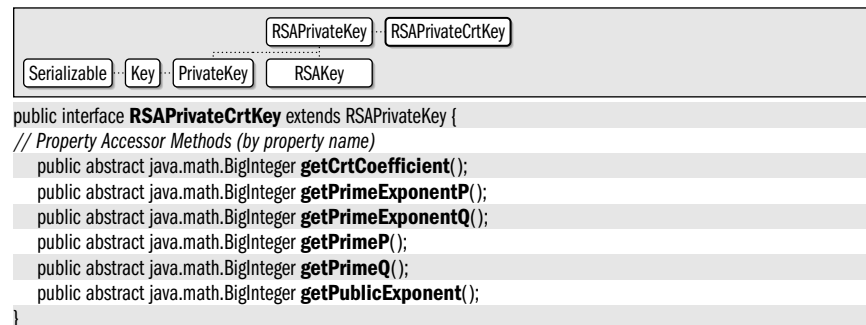
RSAPrivateCrtKey

Java 1.2

java.security.interfaces

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`.



RSAPrivateKey

Java 1.2

java.security.interfaces

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`.



RSAPrivateKey

```
public interface RSAPrivateKey extends java.security.PrivateKey, RSAKey {  
    // Public Instance Methods  
    public abstract java.math.BigInteger getPrivateExponent();  
}
```

Implementations: RSAMultiPrimePrivateCrtKey, RSAPrivateCrtKey

RSAPublicKey

Java 1.2

java.security.interfaces

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`.



```
public interface RSAPublicKey extends java.security.PublicKey, RSAKey {  
    // Public Instance Methods  
    public abstract java.math.BigInteger getPublicExponent();  
}
```

Package java.security.spec

Java 1.2

The `java.security.spec` package contains classes that define transparent representations for DSA and RSA 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 KeySpec;
```

Classes:

```
public class DSAParameterSpec implements AlgorithmParameterSpec,  
    java.security.interfaces.DSAParams;  
public class DSAPrivateKeySpec implements KeySpec;  
public class DSAPublicKeySpec implements KeySpec;  
public abstract class EncodedKeySpec implements KeySpec;  
    public class PKCS8EncodedKeySpec extends EncodedKeySpec;  
    public class X509EncodedKeySpec extends EncodedKeySpec;  
public class PSSParameterSpec implements AlgorithmParameterSpec;  
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 1.2

java.security.spec

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: DSAPrimitiveSpec, PSSParameterSpec, RSAKeyGenParameterSpec, javax.crypto.spec.DHGenParameterSpec, javax.crypto.spec.DHParameterSpec, javax.crypto.spec.IvParameterSpec, 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()

DSAPrimitiveSpec

Java 1.2

java.security.spec

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



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

DSAPrivateKeySpec

Java 1.2

java.security.spec

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



```
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
}
```

DSAPrivateKeySpec

```
public java.math.BigInteger getG();
public java.math.BigInteger getP();
public java.math.BigInteger getQ();
public java.math.BigInteger getX();
}
```

DSAPublicKeySpec

Java 1.2

java.security.spec

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



```
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();
}
```

EncodedKeySpec

Java 1.2

java.security.spec

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



```
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

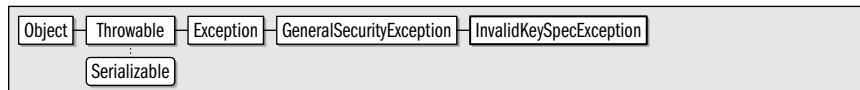
InvalidKeySpecException

Java 1.2

java.security.spec

serializable checked

This exception signals a problem with a KeySpec.



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


PKCS8EncodedKeySpec

Thrown By: java.security.KeyFactory.{generatePrivate(), generatePublic(), getKeySpec()},
java.security.KeyFactorySpi.{engineGeneratePrivate(), engineGeneratePublic(), engineGetKeySpec()},
javax.crypto.EncryptedPrivateKeyInfo.getKeySpec(), javax.crypto.SecretKeyFactory.{generateSecret(),
getKeySpec()}, javax.crypto.SecretKeyFactorySpi.{engineGenerateSecret(), engineGetKeySpec()}

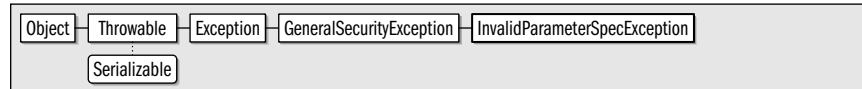
InvalidParameterSpecException

Java 1.2

java.security.spec

serializable checked

This exception signals a problem with an AlgorithmParameterSpec.



```
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 1.2

java.security.spec

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, EncodedKeySpec, RSAPrivateKeySpec, RSAPublicKeySpec, javax.crypto.spec.DESedeKeySpec, javax.crypto.spec.DESKeySpec, javax.crypto.spec.DHPrivateKeySpec, 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()

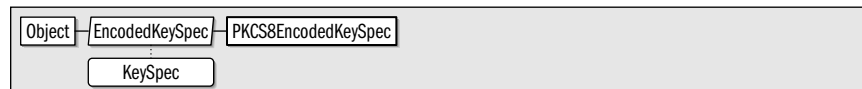
Returned By: java.security.KeyFactory.getKeySpec(),
java.security.KeyFactorySpi.engineGetKeySpec(), javax.crypto.SecretKeyFactory.getKeySpec(),
javax.crypto.SecretKeyFactorySpi.engineGetKeySpec()

PKCS8EncodedKeySpec

Java 1.2

java.security.spec

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



```
public class PKCS8EncodedKeySpec extends EncodedKeySpec {
    // Public Constructors
    public PKCS8EncodedKeySpec(byte[] encodedKey);
}
```

PKCS8EncodedKeySpec

```
// Public Methods Overriding EncodedKeySpec
public byte[] getEncoded();
public final String getFormat();
}
```

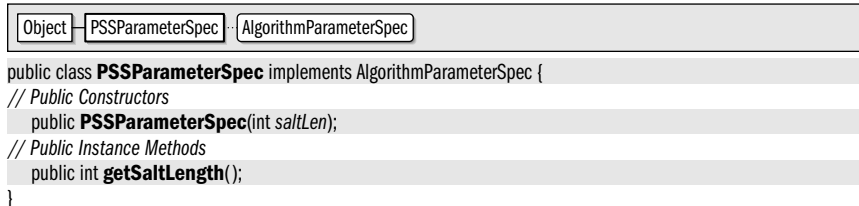
Returned By: javax.crypto.EncryptedPrivateKeyInfo.getKeySpec()

PSSParameterSpec

Java 1.4

java.security.spec

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.

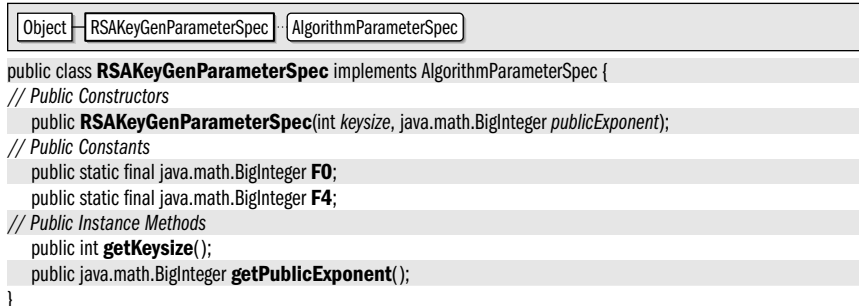


RSAPublicKeyParameterSpec

Java 1.3

java.security.spec

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

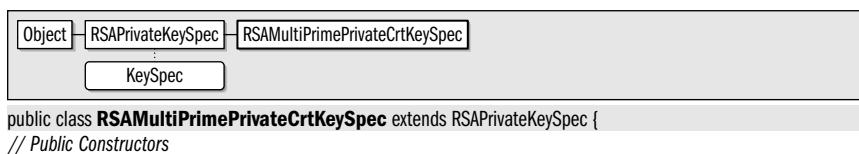


RSAPrivateKeyParameterSpec

Java 1.4

java.security.spec

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.



```

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);

// Property Accessor Methods (by property name)
public java.math.BigInteger getCrtCoefficient();
public RSAOtherPrimeInfo[] getOtherPrimeInfo();
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();
}

```

RSAOtherPrimeInfo

Java 1.4

java.security.spec

This class represents the triplet (prime, exponent, coefficient) that constitutes an “OtherPrimeInfo” structure 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 1.2

java.security.spec

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



```

public class RSAPrivateCrtKeySpec extends RSAPrivateKeySpec {
// Public Constructors
    public RSAPrivateCrtKeySpec(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);

// Property Accessor Methods (by property name)
    public java.math.BigInteger getCrtCoefficient();
    public java.math.BigInteger getPrimeExponentP();
}

```

RSAPrivateCrtKeySpec

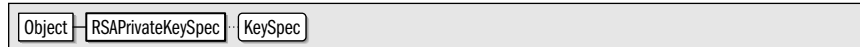
```
public java.math.BigInteger getPrimeExponentQ();
public java.math.BigInteger getPrimeP();
public java.math.BigInteger getPrimeQ();
public java.math.BigInteger getPublicExponent();
}
```

RSAPrivateKeySpec

Java 1.2

java.security.spec

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



```
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 1.2

java.security.spec

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



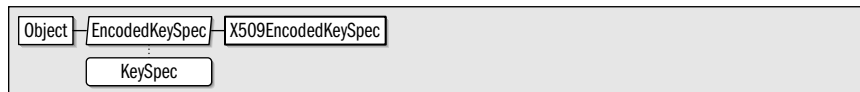
```
public class RSAPublicKeySpec implements KeySpec {
    // Public Constructors
    public RSAPublicKeySpec(java.math.BigInteger modulus, java.math.BigInteger publicExponent);
    // Public Instance Methods
    public java.math.BigInteger getModulus();
    public java.math.BigInteger getPublicExponent();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

X509EncodedKeySpec

Java 1.2

java.security.spec

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



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