http://tutorials.jenkov.com/java-cryptography/keystore.html

**Java KeyStore**

* [Creating a KeyStore](http://tutorials.jenkov.com/java-cryptography/keystore.html#creating-a-keystore)
* [Loading the KeyStore](http://tutorials.jenkov.com/java-cryptography/keystore.html#loading-the-keystore)
* [Getting Keys](http://tutorials.jenkov.com/java-cryptography/keystore.html#getting-keys)
* [Setting Keys](http://tutorials.jenkov.com/java-cryptography/keystore.html#setting-keys)
* [Storing the KeyStore](http://tutorials.jenkov.com/java-cryptography/keystore.html#storing-the-keystore)

|  |  |
| --- | --- |
|  | Jakob Jenkov Last update: 2017-11-30 |

The Java *KeyStore* is a database that can contain keys. A Java KeyStore is represented by the KeyStore (java.security.KeyStore) class. A KeyStore can be written to disk and read again. The KeyStore as a whole can be protected with a password, and each key entry in the KeyStore can be protected with its own password. This makes the KeyStore class a useful mechanism to handle encryption keys securely.

A KeyStore can hold the following types of keys:

* Private keys
* Public keys + certificates
* Secret keys

Private and public keys are used in asymmetric encryption. A public key can have an associated certificate. A certificate is a document that verifies the identity of the person, organization or device claiming to own the public key. A certificate is typically digitally signed by the verifying party as proof.

Secret keys are used in symmetric encryption. In many cases symmetric keys are negotiated when a secure connection is set up. Therefore you will more often be storing public and private keys in a KeyStore than secret keys.

**Creating a KeyStore**

You can create a Java KeyStore instance by calling its getInstance() method. Here is an example of creating a KeyStore instance:

KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());

This example creates a KeyStore instance of Java's default type. It is also possible to create other types of KeyStore instance by passing a different parameter to the getInstance() method. For instance, here is an example that creates a PKCS12 type KeyStore:

KeyStore keyStore = KeyStore.getInstance("PKCS12");

**Loading the KeyStore**

Before a KeyStore instance can be used, it must be loaded. KeyStore instances are often written to disk or other kinds of storage for later use. That is why the KeyStore class assumes that you must read its data in before you can use it. However, it is possible to initialize an empty KeyStore instance with no data, as you will see later.

Loading the KeyStore data from a file or other storage is done by calling the KeyStore load() method. The load() takes two parameters:

1. An [InputStream](http://tutorials.jenkov.com/java-io/inputstream.html) from which to load the KeyStore data.
2. A char[] (char array) containing the KeyStore password.

Here is an example of loading a Java KeyStore:

char[] keyStorePassword = "123abc".toCharArray();

try(InputStream keyStoreData = new FileInputStream("keystore.ks")){

keyStore.load(keyStoreData, keyStorePassword);

}

This example loads the KeyStore file located in the keystore.ks file.

If you don't want to load any data into the KeyStore, just pass null for the InputStream parameter. Here is how loading an empty KeyStore looks:

keyStore3.load(null, keyStorePassword);

You must always load the KeyStore instance, either with data or with null. Otherwise the KeyStore is uninitialized, and all calls to its methods will throw an exception.

**Getting Keys**

You can get the keys of a Java KeyStore instance via its getEntry() method. A KeyStore entry is mapped to an alias which identifies the key, and is protected with a key password. Thus, to access a key you must pass the key alias and password to the getEntry() method. Here is an example of accessing a key entry in a KeyStore instance:

char[] keyPassword = "789xyz".toCharArray();

KeyStore.ProtectionParameter entryPassword =

new KeyStore.PasswordProtection(keyPassword);

KeyStore.Entry keyEntry = keyStore3.getEntry("keyAlias", entryPassword);

If you know that the key entry you want to access is a private key, you can cast the KeyStore.Entry instance to a KeyStore.PrivateKeyEntry. Here is how that looks:

KeyStore.PrivateKeyEntry privateKeyEntry = (KeyStore.PrivateKeyEntry)

keyStore3.getEntry("keyAlias", entryPassword);

After casting to a KeyStore.PrivateKeyEntry you can access the private key, certificate and certificate chain via these methods:

* getPrivateKey()
* getCertificate()
* getCertificateChain()

**Setting Keys**

You can also set keys into a KeyStore instance. Here is an example of setting a secret key (symmetric key) into a KeyStore instance:

SecretKey secretKey = getSecretKey();

KeyStore.SecretKeyEntry secretKeyEntry = new KeyStore.SecretKeyEntry(secretKey);

keyStore3.setEntry("keyAlias2", secretKeyEntry, entryPassword);

**Storing the KeyStore**

Sometimes you may want to store a KeyStore to some storage (disk, database etc.) so you can load it again another time. You store a KeyStore by calling the store() method. Here is an example of storing a KeyStore

char[] keyStorePassword = "123abc".toCharArray();

try (FileOutputStream keyStoreOutputStream = new FileOutputStream("data/keystore.ks")) {

keyStore3.store(keyStoreOutputStream, keyStorePassword);

}