IMiS Base Java Runtime

Imaging Systems Inc.

Package
com.imis

com.imis
Class GlobalizedException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-com.imis.GlobalizedException
```

All Implemented Interfaces:
Serializable

---

public class **GlobalizedException**
extends Exception

Represents globalized errors that occur during application execution.

`GlobalizedException` class extends `java.lang.Exception` class and is used as a base class for exception classes that provide localized errors with the help of the specified resource bundle.

Constructor Summary

`public`	`GlobalizedException(ResourceBundle resources, String resourceName)` Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle` and a name of the string resource that describes the current exception.
`public`	`GlobalizedException(ResourceBundle resources, String resourceName, Object arg0)` Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle` and a name of the formatted string resource that describes the current exception.
`public`	`GlobalizedException(ResourceBundle resources, String resourceName, Object[] args)` Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle` and a name of the formatted string resource that describes the current exception.
`public`	`GlobalizedException(ResourceBundle resources, String resourceName, Throwable cause)` Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle`, a name of the formatted string resource that describes the current exception and a reference to the throwable that is the cause of this exception.
`public`	`GlobalizedException(ResourceBundle resources, String resourceName, Throwable cause, Object arg0)` Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle`, a name of the formatted string resource that describes the current exception and a reference to the throwable that is the cause of this exception.
`public`	`GlobalizedException(ResourceBundle resources, String resourceName, Throwable cause, Object[] args)` Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle`, a name of the formatted string resource that describes the current exception and a reference to the throwable that is the cause of this exception.

Method Summary

`String`	`getMessage()` Gets a message that describes the current exception.

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

GlobalizedException

```
public GlobalizedException(ResourceBundle resources,
                           String resourceName)
```

Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle` and a name of the string resource that describes the current exception.

The `Throwable.getMessage()` of the new instance is initialized with the specified string resource or the resource name, if the string resource is not found in the resources.

Parameters:

`resources` - a `ResourceBundle` with string resources.
`resourceName` - the name of the string resource that describes the current exception.

GlobalizedException

```
public GlobalizedException(ResourceBundle resources,
                           String resourceName,
                           Object arg0)
```

Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle` and a name of the formatted string resource that describes the current exception.

The `Throwable.getMessage()` of the new instance is initialized with the specified formatted string resource or the resource name, if the string resource is not found in the resources.

Parameters:

`resources` - a `ResourceBundle` with string resources.
`resourceName` - the name of the string resource containing one format item.
`arg0` - an object to format.

GlobalizedException

```
public GlobalizedException(ResourceBundle resources,
                           String resourceName,
                           Object[] args)
```

(continued on next page)

(continued from last page)

Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle` and a name of the formatted string resource that describes the current exception.

The `Throwable.getMessage()` of the new instance is initialized with the specified formatted string resource or the resource name, if the string resource is not found in the resources.

Parameters:

`resources` - a `ResourceBundle` with string resources.

`resourceName` - the name of the string resource containing zero or more format items that describes the current exception.

`args` - an object array containing zero or more objects to format.

GlobalizedException

```
public GlobalizedException(ResourceBundle resources,
                           String resourceName,
                           Throwable cause)
```

Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle`, a name of the formatted string resource that describes the current exception and a reference to the throwable that is the cause of this exception.

The `Throwable.getMessage()` and `Throwable.getCause()` of the new instance are initialized with the specified string resource or the resource name, if the string resource is not found in the resources, and a throwable cause respectively.

Parameters:

`resources` - a `ResourceBundle` with string resources.

`resourceName` - the name of the string resource that describes the current exception.

`cause` - the throwable that is the cause of the current exception, or a `null` reference if no cause is specified.

GlobalizedException

```
public GlobalizedException(ResourceBundle resources,
                           String resourceName,
                           Throwable cause,
                           Object arg0)
```

Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle`, a name of the formatted string resource that describes the current exception and a reference to the throwable that is the cause of this exception.

The `Throwable.getMessage()` and `Throwable.getCause()` of the new instance are initialized with the specified string resource or the resource name, if the string resource is not found in the resources and throwable that is the cause of this exception respectively.

Parameters:

`resources` - a `ResourceBundle` with string resources.

`resourceName` - the name of the string resource containing one format item.

`cause` - the throwable that is the cause of the current exception, or a `null` reference if no cause is specified.

`arg0` - an object to format.

GlobalizedException

```
public GlobalizedException(ResourceBundle resources,
                           String resourceName,
                           Throwable cause,
                           Object[] args)
```

(continued on next page)

(continued from last page)

Initializes a new instance of the `GlobalizedException` class with a specified `java.util.ResourceBundle`, a name of the formatted string resource that describes the current exception and a reference to the throwable that is the cause of this exception.

The `Throwable.getMessage()` and `Throwable.getCause()` of the new instance are initialized with the specified string resource or the resource name, if the string resource is not found in the resources and throwable that is the cause of this exception respectively.

Parameters:

`resources` - a `ResourceBundle` with string resources.

`resourceName` - the name of the string resource containing zero or more format items that describes the current exception.

`cause` - the throwable that is the cause of the current exception, or a `null` reference if no cause is specified.

`args` - an object array containing zero or more objects to format.

Methods

getMessage

`public String **getMessage**()`

Gets a message that describes the current exception.

Returns:

A message that describes the current exception (which may be `null`).

com.imis
Class GlobalizedResourceBundle

```
java.lang.Object
  |
  +-java.util.ResourceBundle
      |
      +-com.imis.GlobalizedResourceBundle
```

public class **GlobalizedResourceBundle**
extends ResourceBundle

Represents a globalized resource bundle that contain locale-specific objects.

`GlobalizedResourceBundle` class extends `java.util.ResourceBundle` class and provides the means to change locale used by the resource bundle.

Fields inherited from class `java.util.ResourceBundle`
`parent`

Constructor Summary

`public`	`GlobalizedResourceBundle(String baseName)` Initializes a new instance of the `GlobalizedResourceBundle` class using the specified base name, the default locale, and the caller's class loader.
`public`	`GlobalizedResourceBundle(String baseName, Locale locale)` Initializes a new instance of the `GlobalizedResourceBundle` class using the specified base name and locale, and the caller's class loader.
`public`	`GlobalizedResourceBundle(String baseName, Locale locale, ClassLoader loader)` Initializes a new instance of the `GlobalizedResourceBundle` class using the specified base name, locale, and class loader.

Method Summary

`Enumeration`	`getKeys()` Gets an enumeration of the keys.
`Locale`	`getLocale()` Gets the locale of this resource bundle.
`Object`	`handleGetObject(String key)` Gets an object for the given `key` from this resource bundle.
`void`	`setLocale(Locale locale)` Sets the locale for which a resource bundle is desired.

Methods inherited from class `java.util.ResourceBundle`
`getBundle, getBundle, getBundle, getKeys, getLocale, getObject, getString, getStringArray, handleGetObject, setParent`

Methods inherited from class `java.lang.Object`

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
wait
```

Constructors

GlobalizedResourceBundle

```
public GlobalizedResourceBundle(String baseName)
```

Initializes a new instance of the `GlobalizedResourceBundle` class using the specified base name, the default locale, and the caller's class loader.

Parameters:
`baseName` - the base name of the resource bundle, a fully qualified class name.

GlobalizedResourceBundle

```
public GlobalizedResourceBundle(String baseName,
                                Locale locale)
```

Initializes a new instance of the `GlobalizedResourceBundle` class using the specified base name and locale, and the caller's class loader.

Parameters:
`baseName` - the base name of the resource bundle, a fully qualified class name.
`locale` - the locale for which a resource bundle is desired.

GlobalizedResourceBundle

```
public GlobalizedResourceBundle(String baseName,
                                Locale locale,
                                ClassLoader loader)
```

Initializes a new instance of the `GlobalizedResourceBundle` class using the specified base name, locale, and class loader.

Parameters:
`baseName` - the base name of the resource bundle, a fully qualified class name.
`locale` - the locale for which a resource bundle is desired.
`loader` - the class loader from which to load the resource bundle.

Methods

getLocale

```
public Locale getLocale()
```

Gets the locale of this resource bundle.

Returns:
The locale of this resource bundle.

setLocale

```
public void setLocale(Locale locale)
```

(continued from last page)

Sets the locale for which a resource bundle is desired.

Parameters:
`locale` - the locale for which a resource bundle is desired.

getKeys

`public Enumeration **getKeys**()`

Gets an enumeration of the keys.

Returns:
An enumeration of the keys.

handleGetObject

`protected Object **handleGetObject**(String key)`

Gets an object for the given `key` from this resource bundle.

Returns `null` if this resource bundle does not contain an object for the given `key`.

Parameters:
`key` - the key for the desired object.

Returns:
The object for the given key, or `null`.

com.imis
Interface IAutoCloseable

public interface **IAutoCloseable**
extends

A resource that must be closed when it is no longer needed.

Note: This interface is equivalent to `AutoCloseable` interface in JRE 7 and should only be used with JRE prior to JRE 7.

Method Summary

`void`	`close()` Closes this resource, relinquishing any underlying resources.

Methods

close

```
public void close()
  throws Exception
```

Closes this resource, relinquishing any underlying resources.

While this interface method is declared to throw `Exception`, implementers are strongly encouraged to declare concrete implementations of the `close()` method to throw more specific exceptions, or to throw no exception at all if the close operation cannot fail.

Implementers of this interface are also strongly advised to not have the `close()` method throw `java.lang.InterruptedException`. This exception interacts with a thread's interrupted status, and runtime misbehavior is likely to occur if an `InterruptedException` is suppressed. More generally, if it would cause problems for an exception to be suppressed, the AutoCloseable.close method should not throw it.

Note that unlike the `close()` method of `java.io.Closeable`, this close method is not required to be idempotent. In other words, calling this close method more than once may have some visible side effect, unlike `Closeable.close` which is required to have no effect if called more than once. However, implementers of this interface are strongly encouraged to make their close methods idempotent.

Throws:
`Exception` - if this resource cannot be closed

Package com.imis.annotation

com.imis.annotation
Interface Flags

public interface **Flags**
extends Annotation

Indicates that an enumeration can be treated as a set of flags.

Methods inherited from interface `java.lang.annotation.Annotation`
`annotationType, equals, hashCode, toString`

com.imis.annotation
Interface Internal

public interface **Internal**
extends Annotation

Indicates that a type or type member is accessible only within a Java archive file and should not be documented for outside use.

Methods inherited from interface `java.lang.annotation.Annotation`
`annotationType, equals, hashCode, toString`

com.imis.annotation
Interface NotNull

public interface **NotNull**
extends Annotation

Indicates that annotated element can never be `null`.

Methods inherited from interface `java.lang.annotation.Annotation`
`annotationType, equals, hashCode, toString`

com.imis.annotation
Interface Nullable

public interface **Nullable**
extends Annotation

Indicates that annotated element can be `null` under some circumstance.

Methods inherited from interface `java.lang.annotation.Annotation`
`annotationType, equals, hashCode, toString`

Package
com.imis.crypto

com.imis.crypto
Class Cipher

```
java.lang.Object
  |
  +-com.imis.crypto.Cipher
```

public class **Cipher**
extends Object

This class provides the functionality of a cryptographic cipher for encryption and decryption. It forms the core of the Imaging Systems Java Cryptographic Extension (JCE) framework.

Currently the only implemented cipher is Square cipher.

In order to create a Cipher object, the application calls the Cipher's `getInstance` method, and passes the name of the requested *transformation* to it. Optionally, the name of a provider may be specified.

A *transformation* is a string that describes the operation (or set of operations) to be performed on the given input, to produce some output. A transformation always includes the name of a cryptographic algorithm (e.g., *Square*), and may be followed by a feedback mode and padding scheme.

A transformation is of the form:

- "*algorithm/mode/padding*" or
- "*algorithm*"

(in the latter case, provider-specific default values for the mode and padding scheme are used). For example, the following is a valid transformation:

```
Cipher c = Cipher.getInstance("Square/CBC/Zeros");
```

See Also:
`javax.crypto.KeyGenerator`, `javax.crypto.SecretKey`
Author:
Robert Petek

Field Summary

`public static final`	`DECRYPT_MODE` Constant used to initialize cipher to decryption mode. Value: **2**
`public static final`	`ENCRYPT_MODE` Constant used to initialize cipher to encryption mode. Value: **1**
`public static final`	`PRIVATE_KEY` Constant used to indicate the key to be unwrapped is a private key. Value: **2**
`public static final`	`PUBLIC_KEY` Constant used to indicate the key to be unwrapped is a public key. Value: **1**

public static final	SECRET_KEY Constant used to indicate the key to be unwrapped is a secret key. Value: **3**
public static final	UNWRAP_MODE Constant used to initialize cipher to key-unwrapping mode. Value: **4**
public static final	WRAP_MODE Constant used to initialize cipher to key-wrapping mode. Value: **3**

Constructor Summary

protected	`Cipher(CipherSpi cipherSpi, java.security.Provider provider, String transformation)` Initializes a new instance of the `Cipher` class.

Method Summary

byte[]	`doFinal()` Finishes a multiple-part encryption or decryption operation, depending on how this cipher was initialized.
byte[]	`doFinal(byte[] input)` Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation.
int	`doFinal(byte[] output, int outputOffset)` Finishes a multiple-part encryption or decryption operation, depending on how this cipher was initialized.
byte[]	`doFinal(byte[] input, int inputOffset, int inputLength)` Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation.
int	`doFinal(byte[] input, int inputOffset, int inputLength, byte[] output)` Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation.
int	`doFinal(byte[] input, int inputOffset, int inputLength, byte[] output, int outputOffset)` Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation.
String	`getAlgorithm()` Returns the algorithm name of this `Cipher` object.
int	`getBlockSize()` Returns the block size, in bytes.
javax.crypto.ExemptionMechanism	`getExemptionMechanism()` Returns the exemption mechanism object used with this cipher.
static Cipher	`getInstance(String transformation)` Creates a new cipher instance for the given transformation.
static Cipher	`getInstance(String transformation, java.security.Provider provider)` Creates a new cipher instance for the given transform and the given provider.
static Cipher	`getInstance(String transformation, String provider)` Creates a new cipher instance for the given transformation and the named provider.

`byte[]`	`getIV()` Returns the initialization vector (IV) in a new buffer.
`int`	`getOutputSize(int inputLength)` Returns the length, in bytes, that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLength`, in bytes.
`java.security.AlgorithmParameters`	`getParameters()` Returns the `java.security.AlgorithmParameters` used with this cipher.
`java.security.Provider`	`getProvider()` Returns the provider of this `Cipher` object.
`void`	`init(int opmode, java.security.cert.Certificate certificate)` Initializes this cipher with the public key from the given certificate.
`void`	`init(int opmode, java.security.cert.Certificate certificate, java.security.SecureRandom random)` Initializes this cipher with the public key from the given certificate and a source of randomness.
`void`	`init(int opmode, java.security.Key key)` Initializes this cipher with a key.
`void`	`init(int opmode, java.security.Key key, java.security.AlgorithmParameters params)` Initializes this cipher with a key and a set of algorithm parameters.
`void`	`init(int opmode, java.security.Key key, java.security.spec.AlgorithmParameterSpec params)` Initializes this cipher with a key and a set of algorithm parameters.
`void`	`init(int opmode, java.security.Key key, java.security.spec.AlgorithmParameterSpec params, java.security.SecureRandom random)` Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.
`void`	`init(int opmode, java.security.Key key, java.security.AlgorithmParameters params, java.security.SecureRandom random)` Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.
`void`	`init(int opmode, java.security.Key key, java.security.SecureRandom random)` Initializes this cipher with a key and a source of randomness.
`java.security.Key`	`unwrap(byte[] wrappedKey, String wrappedKeyAlgorithm, int wrappedKeyType)` Unwraps a previously wrapped key.
`byte[]`	`update(byte[] input)` Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.
`byte[]`	`update(byte[] input, int inputOffset, int inputLength)` Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.
`int`	`update(byte[] input, int inputOffset, int inputLength, byte[] output)` Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

`int`	`update(byte[] input, int inputOffset, int inputLength, byte[] output, int outputOffset)` Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.
`byte[]`	`wrap(java.security.Key key)` Wraps a key.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

DECRYPT_MODE

`public static final int` **`DECRYPT_MODE`**

Constant used to initialize cipher to decryption mode.
Constant value: **`2`**

ENCRYPT_MODE

`public static final int` **`ENCRYPT_MODE`**

Constant used to initialize cipher to encryption mode.
Constant value: **`1`**

PRIVATE_KEY

`public static final int` **`PRIVATE_KEY`**

Constant used to indicate the key to be unwrapped is a private key.
Constant value: **`2`**

PUBLIC_KEY

`public static final int` **`PUBLIC_KEY`**

Constant used to indicate the key to be unwrapped is a public key.
Constant value: **`1`**

SECRET_KEY

`public static final int` **`SECRET_KEY`**

Constant used to indicate the key to be unwrapped is a secret key.
Constant value: **`3`**

UNWRAP_MODE

`public static final int` **`UNWRAP_MODE`**

Constant used to initialize cipher to key-unwrapping mode.
Constant value: **`4`**

WRAP_MODE

```
public static final int WRAP_MODE
```

Constant used to initialize cipher to key-wrapping mode.
Constant value: **3**

Constructors

Cipher

```
protected Cipher(CipherSpi cipherSpi,
                 java.security.Provider provider,
                 String transformation)
```

Initializes a new instance of the `Cipher` class.

Parameters:

`cipherSpi` - the underlying implementation of the cipher.
`provider` - the provider of this cipher implementation.
`transformation` - the transformation this cipher performs.

Methods

getInstance

```
public static Cipher getInstance(String transformation)
  throws java.security.NoSuchAlgorithmException,
         javax.crypto.NoSuchPaddingException
```

Creates a new cipher instance for the given transformation.

The installed providers are tried in order for an implementation, and the first appropriate instance is returned. If no installed provider can provide the implementation, an appropriate exception is thrown.

Parameters:

`transformation` - The transformation to create.

Returns:

An appropriate cipher for this transformation.

Throws:

`java.security.NoSuchAlgorithmException` - If no installed provider can supply the appropriate cipher or mode.
`javax.crypto.NoSuchPaddingException` - If no installed provider can supply the appropriate padding.

getInstance

```
public static Cipher getInstance(String transformation,
         String provider)
  throws java.security.NoSuchAlgorithmException,
         java.security.NoSuchProviderException,
         javax.crypto.NoSuchPaddingException
```

(continued from last page)

Creates a new cipher instance for the given transformation and the named provider.

Parameters:
`transformation` - The transformation to create.
`provider` - The name of the provider to use.

Returns:
An appropriate cipher for this transformation.

Throws:
`java.security.NoSuchAlgorithmException` - If the provider cannot supply the appropriate cipher or mode.
`java.security.NoSuchProviderException` - If the named provider is not installed.
`javax.crypto.NoSuchPaddingException` - If the provider cannot supply the appropriate padding.

getInstance

```
public static Cipher getInstance(String transformation,
         java.security.Provider provider)
  throws java.security.NoSuchAlgorithmException,
         javax.crypto.NoSuchPaddingException
```

Creates a new cipher instance for the given transform and the given provider.

Parameters:
`transformation` - The transformation to create.
`provider` - The provider to use.

Returns:
An appropriate cipher for this transformation.

Throws:
`java.security.NoSuchAlgorithmException` - If the given provider cannot supply the appropriate cipher or mode.
`javax.crypto.NoSuchPaddingException` - If the given provider cannot supply the appropriate padding scheme.

getProvider

```
public final java.security.Provider getProvider()
```

Returns the provider of this `Cipher` object.

Returns:
The provider of this `Cipher` object.

getAlgorithm

```
public final String getAlgorithm()
```

Returns the algorithm name of this `Cipher` object.

This equals to the `transformation` parameter that was specified in one of the `getInstance` calls that created this `Cipher` object.

Returns:
The algorithm name of this `Cipher` object.

(continued from last page)

getBlockSize

`public final int getBlockSize()`

Returns the block size, in bytes.

Returns:
The block size, in bytes, or 0 if the underlying algorithm is not a block cipher.

getIV

`public final byte[] getIV()`

Returns the initialization vector (IV) in a new buffer.

This is useful in the case where a random IV was created, or in the context of password-based encryption or decryption, where the IV is derived from a user-supplied password.

Returns:
The initialization vector in a new buffer, or `null` if the underlying algorithm does not use an IV, or if the IV has not yet been set.

getParameters

`public final java.security.AlgorithmParameters getParameters()`

Returns the `java.security.AlgorithmParameters` used with this cipher.

The returned parameters may be the same that were used to initialize this cipher, or may contain a combination of default and random parameter values used by the underlying cipher implementation if this cipher requires algorithm parameters but was not initialized with any.

Returns:
The parameters used with this cipher, or `null` if this cipher does not use any parameters.

getOutputSize

```
public final int getOutputSize(int inputLength)
  throws IllegalStateException
```

Returns the length, in bytes, that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLength`, in bytes.

This call takes into account any unprocessed (buffered) data from a previous `update` call, and padding.

The actual output length of the next `update` or `doFinal` call may be smaller than the length returned by this method.

Parameters:
`inputLength` - the input length, in bytes.

Returns:
The required output buffer size, in bytes.

Throws:
`IllegalStateException` - if this cipher is in a wrong state (e.g., has not yet been initialized).

getExemptionMechanism

`public final javax.crypto.ExemptionMechanism getExemptionMechanism()`

(continued from last page)

Returns the exemption mechanism object used with this cipher.

This method currently always returns `null`.

Returns:
The exemption mechanism object used with this cipher, or `null` if none used.

init

```
public final void init(int opmode,
         java.security.Key key)
  throws java.security.InvalidKeyException
```

Initializes this cipher with a key.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of `opmode`.

If this cipher requires any algorithm parameters that cannot be derived from the given `key`, the underlying cipher implementation is supposed to generate the required parameters itself (using provider specific default or random values) if it is being initialized for encryption or key wrapping, and raise an `InvalidKeyException` if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using `getParameters` or `getIV` (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them using the `SecureRandom` implementation of the highest-priority installed provider as the source of randomness. If none of the installed providers supply an implementation of SecureRandom, a system-provided source of randomness will be used.

Note that when a Cipher object is initialized, it loses all previously acquired state. In other words, initializing a `Cipher` is equivalent to creating a new instance of that `Cipher` and initializing it.

Parameters:
`opmode` - the operation mode of this cipher (this is one of the following: `ENCRYPT_MODE`, `DECRYPT_MODE`, `WRAP_MODE` or `UNWRAP_MODE`)
`key` - the key

Throws:
`InvalidKeyException` - if the given key is inappropriate for initializing this cipher, or if this cipher is being initialized for decryption and requires algorithm parameters that cannot be determined from the given key, or if the given key has a keysize that exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files).

init

```
public final void init(int opmode,
         java.security.Key key,
         java.security.SecureRandom random)
  throws java.security.InvalidKeyException
```

Initializes this cipher with a key and a source of randomness.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of `opmode`.

If this cipher requires any algorithm parameters that cannot be derived from the given `key`, the underlying cipher implementation is supposed to generate the required parameters itself (using provider specific default or random values) if it is being initialized for encryption or key wrapping, and raise an `InvalidKeyException` if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using `getParameters` or `getIV` (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them from `random`.

Note that when a Cipher object is initialized, it loses all previously acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

(continued from last page)

Parameters:

`opmode` - the operation mode of this cipher (this is one of the following: `ENCRYPT_MODE`, `DECRYPT_MODE`, `WRAP_MODE` or `UNWRAP_MODE`)
`key` - the encryption key
`random` - the source of randomness

Throws:

`InvalidKeyException` - if the given key is inappropriate for initializing this cipher, or if this cipher is being initialized for decryption and requires algorithm parameters that cannot be determined from the given key, or if the given key has a keysize that exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files).

---

init

```
public final void init(int opmode,
         java.security.Key key,
         java.security.spec.AlgorithmParameterSpec params)
  throws java.security.InvalidKeyException,
         java.security.InvalidAlgorithmParameterException
```

Initializes this cipher with a key and a set of algorithm parameters.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of `opmode`.

If this cipher requires any algorithm parameters and `params` is null, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an `InvalidAlgorithmParameterException` if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using `getParameters` or `getIV` (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them using the `SecureRandom` implementation of the highest-priority installed provider as the source of randomness. (If none of the installed providers supply an implementation of SecureRandom, a system-provided source of randomness will be used.)

Note that when a Cipher object is initialized, it loses all previously acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

Parameters:

`opmode` - the operation mode of this cipher (this is one of the following: `ENCRYPT_MODE`, `DECRYPT_MODE`, `WRAP_MODE` or `UNWRAP_MODE`).
`key` - the encryption key.
`params` - the algorithm parameters.

Throws:

`InvalidKeyException` - if the given key is inappropriate for initializing this cipher, or its keysize exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files).
`InvalidAlgorithmParameterException` - if the given algorithm parameters are inappropriate for this cipher, or this cipher is being initialized for decryption and requires algorithm parameters and `params` is null, or the given algorithm parameters imply a cryptographic strength that would exceed the legal limits (as determined from the configured jurisdiction policy files).

---

init

```
public final void init(int opmode,
         java.security.Key key,
         java.security.spec.AlgorithmParameterSpec params,
         java.security.SecureRandom random)
  throws java.security.InvalidKeyException,
         java.security.InvalidAlgorithmParameterException
```

(continued from last page)

Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of `opmode`.

If this cipher requires any algorithm parameters and `params` is `null`, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an `InvalidAlgorithmParameterException` if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using `getParameters` or `getIV` (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them from `random`.

Note that when a Cipher object is initialized, it loses all previously acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

Parameters:

`opmode` - the operation mode of this cipher (this is one of the following: `ENCRYPT_MODE`, `DECRYPT_MODE`, `WRAP_MODE` or `UNWRAP_MODE`)
`key` - the encryption key
`params` - the algorithm parameters
`random` - the source of randomness

Throws:

`InvalidKeyException` - if the given key is inappropriate for initializing this cipher, or its keysize exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files).
`InvalidAlgorithmParameterException` - if the given algorithm parameters are inappropriate for this cipher, or this cipher is being initialized for decryption and requires algorithm parameters and `params` is null, or the given algorithm parameters imply a cryptographic strength that would exceed the legal limits (as determined from the configured jurisdiction policy files).

init

```
public final void init(int opmode,
         java.security.Key key,
         java.security.AlgorithmParameters params)
  throws java.security.InvalidKeyException,
         java.security.InvalidAlgorithmParameterException
```

Initializes this cipher with a key and a set of algorithm parameters.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of `opmode`.

If this cipher requires any algorithm parameters and `params` is null, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an `InvalidAlgorithmParameterException` if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using `getParameters` or `getIV` (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them using the `SecureRandom` implementation of the highest-priority installed provider as the source of randomness. (If none of the installed providers supply an implementation of SecureRandom, a system-provided source of randomness will be used.)

Note that when a Cipher object is initialized, it loses all previously acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

Parameters:

`opmode` - the operation mode of this cipher (this is one of the following: `ENCRYPT_MODE`, `DECRYPT_MODE`, `WRAP_MODE` or `UNWRAP_MODE`).
`key` - the encryption key.
`params` - the algorithm parameters.

Throws:

(continued from last page)

`InvalidKeyException` - if the given key is inappropriate for initializing this cipher, or its keysize exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files).
`InvalidAlgorithmParameterException` - if the given algorithm parameters are inappropriate for this cipher, or this cipher is being initialized for decryption and requires algorithm parameters and `params` is null, or the given algorithm parameters imply a cryptographic strength that would exceed the legal limits (as determined from the configured jurisdiction policy files).

init

```
public final void init(int opmode,
         java.security.Key key,
         java.security.AlgorithmParameters params,
         java.security.SecureRandom random)
  throws java.security.InvalidKeyException,
         java.security.InvalidAlgorithmParameterException
```

Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of `opmode`.

If this cipher requires any algorithm parameters and `params` is null, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an `InvalidAlgorithmParameterException` if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using `getParameters` or `getIV` (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them from `random`.

Note that when a Cipher object is initialized, it loses all previously acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

Parameters:

`opmode` - the operation mode of this cipher (this is one of the following: `ENCRYPT_MODE`, `DECRYPT_MODE`, `WRAP_MODE` or `UNWRAP_MODE`).
`key` - the encryption key.
`params` - the algorithm parameters.
`random` - the source of randomness.

Throws:

`InvalidKeyException` - if the given key is inappropriate for initializing this cipher, or its keysize exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files).
`InvalidAlgorithmParameterException` - if the given algorithm parameters are inappropriate for this cipher, or this cipher is being initialized for decryption and requires algorithm parameters and `params` is null, or the given algorithm parameters imply a cryptographic strength that would exceed the legal limits (as determined from the configured jurisdiction policy files).

init

```
public final void init(int opmode,
         java.security.cert.Certificate certificate)
  throws java.security.InvalidKeyException
```

(continued from last page)

Initializes this cipher with the public key from the given certificate.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of `opmode`.

If the certificate is of type X.509 and has a *key usage* extension field marked as critical, and the value of the *key usage* extension field implies that the public key in the certificate and its corresponding private key are not supposed to be used for the operation represented by the value of `opmode`, an `InvalidKeyException` is thrown.

If this cipher requires any algorithm parameters that cannot be derived from the public key in the given certificate, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an `InvalidKeyException` if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using `getParameters` or `getIV` (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them using the `SecureRandom` implementation of the highest-priority installed provider as the source of randomness. (If none of the installed providers supply an implementation of SecureRandom, a system-provided source of randomness will be used.)

Note that when a Cipher object is initialized, it loses all previously acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

Parameters:

`opmode` - the operation mode of this cipher (this is one of the following: `ENCRYPT_MODE`, `DECRYPT_MODE`, `WRAP_MODE` or `UNWRAP_MODE`)
`certificate` - the certificate

Throws:

`InvalidKeyException` - if the public key in the given certificate is inappropriate for initializing this cipher, or this cipher is being initialized for decryption or unwrapping keys and requires algorithm parameters that cannot be determined from the public key in the given certificate, or the keysize of the public key in the given certificate has a keysize that exceeds the maximum allowable keysize (as determined by the configured jurisdiction policy files).

init

```
public final void init(int opmode,
          java.security.cert.Certificate certificate,
          java.security.SecureRandom random)
  throws java.security.InvalidKeyException
```

Initializes this cipher with the public key from the given certificate and a source of randomness.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of `opmode`.

If the certificate is of type X.509 and has a *key usage* extension field marked as critical, and the value of the *key usage* extension field implies that the public key in the certificate and its corresponding private key are not supposed to be used for the operation represented by the value of `opmode`, an `InvalidKeyException` is thrown.

If this cipher requires any algorithm parameters that cannot be derived from the public key in the given `certificate`, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an `InvalidKeyException` if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using `getParameters` or `getIV` (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them from `random`.

Note that when a Cipher object is initialized, it loses all previously acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

Parameters:

`opmode` - the operation mode of this cipher (this is one of the following: `ENCRYPT_MODE`, `DECRYPT_MODE`, `WRAP_MODE` or `UNWRAP_MODE`).
`certificate` - the certificate.
`random` - the source of randomness.

(continued from last page)

Throws:

`InvalidKeyException` - if the public key in the given certificate is inappropriate for initializing this cipher, or this cipher is being initialized for decryption or unwrapping keys and requires algorithm parameters that cannot be determined from the public key in the given certificate, or the keysize of the public key in the given certificate has a keysize that exceeds the maximum allowable keysize (as determined by the configured jurisdiction policy files).

update

```
public final byte[] update(byte[] input)
  throws IllegalStateException
```

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

The bytes in the `input` buffer are processed, and the result is stored in a new buffer.

If `input` has a length of zero, this method returns `null`.

Parameters:

`input` - the input buffer.

Returns:

The new buffer with the result, or `null` if the underlying cipher is a block cipher and the input data is too short to result in a new block.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized)

update

```
public final byte[] update(byte[] input,
          int inputOffset,
          int inputLength)
  throws IllegalStateException
```

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

The first `inputLen` bytes in the `input` buffer, starting at `inputOffset` inclusive, are processed, and the result is stored in a new buffer.

If `inputLen` is zero, this method returns `null`.

Parameters:

`input` - the input buffer.

`inputOffset` - the offset in `input` where the input starts.

`inputLength` - the input length.

Returns:

The new buffer with the result, or `null` if the underlying cipher is a block cipher and the input data is too short to result in a new block.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized)

(continued from last page)

update

```
public final int update(byte[] input,
                 int inputOffset,
                 int inputLength,
                 byte[] output)
    throws IllegalStateException,
           javax.crypto.ShortBufferException
```

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

The first `inputLen` bytes in the `input` buffer, starting at `inputOffset` inclusive, are processed, and the result is stored in the `output` buffer.

If the `output` buffer is too small to hold the result, a `ShortBufferException` is thrown. In this case, repeat this call with a larger output buffer. Use `getOutputSize` to determine how big the output buffer should be.

If `inputLen` is zero, this method returns a length of zero.

Note: this method should be copy-safe, which means the `input` and `output` buffers can reference the same byte array and no unprocessed input data is overwritten when the result is copied into the output buffer.

Parameters:

`input` - the input buffer.
`inputOffset` - the offset in `input` where the input starts.
`inputLength` - the input length.
`output` - the buffer for the result.

Returns:

The number of bytes stored in `output`.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized)
`ShortBufferException` - if the given output buffer is too small to hold the result.

update

```
public final int update(byte[] input,
                 int inputOffset,
                 int inputLength,
                 byte[] output,
                 int outputOffset)
    throws IllegalStateException,
           javax.crypto.ShortBufferException
```

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

The first `inputLen` bytes in the `input` buffer, starting at `inputOffset` inclusive, are processed, and the result is stored in the `output` buffer, starting at `outputOffset` inclusive.

If the `output` buffer is too small to hold the result, a `ShortBufferException` is thrown. In this case, repeat this call with a larger output buffer. Use `getOutputSize` to determine how big the output buffer should be.

If `inputLen` is zero, this method returns a length of zero.

Note: this method should be copy-safe, which means the `input` and `output` buffers can reference the same byte array and no unprocessed input data is overwritten when the result is copied into the output buffer.

Parameters:

`input` - the input buffer.
`inputOffset` - the offset in `input` where the input starts.
`inputLength` - the input length.
`output` - the buffer for the result.

(continued from last page)

`outputOffset` - the offset in `output` where the result is stored.

Returns:

The number of bytes stored in `output`.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized)

`ShortBufferException` - if the given output buffer is too small to hold the result.

---

doFinal

```
public final byte[] doFinal()
  throws IllegalStateException,
         javax.crypto.IllegalBlockSizeException,
         javax.crypto.BadPaddingException
```

Finishes a multiple-part encryption or decryption operation, depending on how this cipher was initialized.

Input data that may have been buffered during a previous `update` operation is processed, with padding (if requested) being applied. The result is stored in a new buffer.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to `init`. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to `init`) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

Returns:

The new buffer with the result.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized)

`IllegalBlockSizeException` - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided.

`BadPaddingException` - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes

---

doFinal

```
public final byte[] doFinal(byte[] input)
  throws IllegalStateException,
         javax.crypto.IllegalBlockSizeException,
         javax.crypto.BadPaddingException
```

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

The bytes in the `input` buffer, and any input bytes that may have been buffered during a previous `update` operation, are processed, with padding (if requested) being applied. The result is stored in a new buffer.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to `init`. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to `init`) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

Parameters:

`input` - the input buffer.

Returns:

The new buffer with the result.

Throws:

(continued from last page)

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized).
`IllegalBlockSizeException` - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided.
`BadPaddingException` - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes

doFinal

```
public final byte[] doFinal(byte[] input,
         int inputOffset,
         int inputLength)
  throws IllegalStateException,
         javax.crypto.IllegalBlockSizeException,
         javax.crypto.BadPaddingException
```

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

The first `inputLen` bytes in the `input` buffer, starting at `inputOffset` inclusive, and any input bytes that may have been buffered during a previous `update` operation, are processed, with padding (if requested) being applied. The result is stored in a new buffer.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to `init`. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to `init`) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

Parameters:

`input` - the input buffer.
`inputOffset` - the offset in `input` where the input starts.
`inputLength` - the input length.

Returns:

The new buffer with the result

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized)
`IllegalBlockSizeException` - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided.
`BadPaddingException` - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes

doFinal

```
public final int doFinal(byte[] output,
         int outputOffset)
  throws IllegalStateException,
         javax.crypto.IllegalBlockSizeException,
         javax.crypto.BadPaddingException,
         javax.crypto.ShortBufferException
```

(continued from last page)

Finishes a multiple-part encryption or decryption operation, depending on how this cipher was initialized.

Input data that may have been buffered during a previous `update` operation is processed, with padding (if requested) being applied. The result is stored in the `output` buffer, starting at `outputOffset` inclusive.

If the `output` buffer is too small to hold the result, a `ShortBufferException` is thrown. In this case, repeat this call with a larger output buffer. Use `getOutputSize` to determine how big the output buffer should be.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to `init`. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to `init`) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

Parameters:

`output` - the buffer for the result.

`outputOffset` - the offset in `output` where the result is stored.

Returns:

The number of bytes stored in `output`.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized)

`IllegalBlockSizeException` - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided.

`ShortBufferException` - if the given output buffer is too small to hold the result.

`BadPaddingException` - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes.

doFinal

```
public final int doFinal(byte[] input,
         int inputOffset,
         int inputLength,
         byte[] output)
  throws IllegalStateException,
         javax.crypto.IllegalBlockSizeException,
         javax.crypto.BadPaddingException,
         javax.crypto.ShortBufferException
```

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

The first `inputLen` bytes in the `input` buffer, starting at `inputOffset` inclusive, and any input bytes that may have been buffered during a previous `update` operation, are processed, with padding (if requested) being applied. The result is stored in the `output` buffer.

If the `output` buffer is too small to hold the result, a `ShortBufferException` is thrown. In this case, repeat this call with a larger output buffer. Use `getOutputSize` to determine how big the output buffer should be.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to `init`. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to `init`) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

Note: this method should be copy-safe, which means the `input` and `output` buffers can reference the same byte array and no unprocessed input data is overwritten when the result is copied into the output buffer.

Parameters:

`input` - the input buffer.

`inputOffset` - the offset in `input` where the input starts.

`inputLength` - the input length.

`output` - the buffer for the result.

(continued from last page)

Returns:

The number of bytes stored in `output`.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized).

`IllegalBlockSizeException` - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided.

`ShortBufferException` - if the given output buffer is too small to hold the result.

`BadPaddingException` - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes.

doFinal

```
public final int doFinal(byte[] input,
         int inputOffset,
         int inputLength,
         byte[] output,
         int outputOffset)
  throws IllegalStateException,
         javax.crypto.IllegalBlockSizeException,
         javax.crypto.BadPaddingException,
         javax.crypto.ShortBufferException
```

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

The first `inputLen` bytes in the `input` buffer, starting at `inputOffset` inclusive, and any input bytes that may have been buffered during a previous `update` operation, are processed, with padding (if requested) being applied. The result is stored in the `output` buffer, starting at `outputOffset` inclusive.

If the `output` buffer is too small to hold the result, a `ShortBufferException` is thrown. In this case, repeat this call with a larger output buffer. Use `getOutputSize` to determine how big the output buffer should be.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to `init`. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to `init`) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

Note: this method should be copy-safe, which means the `input` and `output` buffers can reference the same byte array and no unprocessed input data is overwritten when the result is copied into the output buffer.

Parameters:

`input` - the input buffer.

`inputOffset` - the offset in `input` where the input starts.

`inputLength` - the input length.

`output` - the buffer for the result.

`outputOffset` - the offset in `output` where the result is stored.

Returns:

The number of bytes stored in `output`.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized)

`IllegalBlockSizeException` - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided.

`ShortBufferException` - if the given output buffer is too small to hold the result.

`BadPaddingException` - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes.

(continued from last page)

wrap

```
public final byte[] wrap(java.security.Key key)
  throws IllegalStateException,
         javax.crypto.IllegalBlockSizeException,
         java.security.InvalidKeyException
```

Wraps a key.

Parameters:

`key` - the key to be wrapped.

Returns:

The wrapped key or a `null` reference if there is no underlying implementation of the cipher.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized).

`IllegalBlockSizeException` - if this cipher is a block cipher, no padding has been requested, and the length of the encoding of the key to be wrapped is not a multiple of the block size.

`InvalidKeyException` - if it is impossible or unsafe to wrap the key with this cipher (e.g., a hardware protected key is being passed to a software-only cipher).

unwrap

```
public final java.security.Key unwrap(byte[] wrappedKey,
         String wrappedKeyAlgorithm,
         int wrappedKeyType)
  throws IllegalStateException,
         java.security.InvalidKeyException,
         java.security.NoSuchAlgorithmException
```

Unwraps a previously wrapped key.

Parameters:

`wrappedKey` - the key to be unwrapped.

`wrappedKeyAlgorithm` - the algorithm associated with the wrapped key.

`wrappedKeyType` - the type of the wrapped key. This must be one of `SECRET_KEY`, `PRIVATE_KEY`, or `PUBLIC_KEY`.

Returns:

The unwrapped key or a `null` reference if there is no underlying implementation of the cipher.

Throws:

`IllegalStateException` - if this cipher is in a wrong state (e.g., has not been initialized).

`NoSuchAlgorithmException` - if no installed providers can create keys of type `wrappedKeyType` for the `wrappedKeyAlgorithm`.

`InvalidKeyException` - if `wrappedKey` does not represent a wrapped key of type `wrappedKeyType` for the `wrappedKeyAlgorithm`.

com.imis.crypto
Class SquareKey

```
java.lang.Object
  |
  +-com.imis.crypto.SquareKey
```

All Implemented Interfaces:
javax.crypto.SecretKey

public class **SquareKey**
extends Object
implements javax.crypto.SecretKey

A secret (symmetric) key for Square cipher.
Author:
Robert Petek

Fields inherited from interface `javax.crypto.SecretKey`
`serialVersionUID`

Fields inherited from interface `java.security.Key`
`serialVersionUID`

Constructor Summary

`public`	`SquareKey(byte[] key)` Initializes a new instance of the `SquareKey` class.
`public`	`SquareKey(byte[] key, int keyOffset)` Initializes a new instance of the `SquareKey` class.

Method Summary

`String`	`getAlgorithm()` Returns the name of the algorithm associated with this Square key.
`byte[]`	`getEncoded()` Returns the key material of this Square key.
`String`	`getFormat()` Returns the name of the encoding format for this Square key.

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.security.Key`
`getAlgorithm, getEncoded, getFormat`

Constructors

SquareKey

```
public SquareKey(byte[] key)
```

Initializes a new instance of the `SquareKey` class.

Creates a `SquareKey` object using the first 16 bytes in `key` as the key material for the Square key.

Parameters:
`key` - the buffer with the Square key material.

Throws:
`NullPointerException` - if `key` is a `null` reference.
`IllegalArgumentException` - if `key` length is less than 16.

SquareKey

```
public SquareKey(byte[] key,
                 int keyOffset)
```

Initializes a new instance of the `SquareKey` class.

Creates a `SquareKey` object using the first 16 bytes in `key`, beginning at `offset` inclusive, as the key material for the Square key.

The bytes that constitute the Square key are those between `key[keyOffset]` and `key[keyOffset+keySize-1]` inclusive.

Parameters:
`key` - the buffer with the Square key material.
`keyOffset` - the offset in `key`, where the Square key material starts.

Throws:
`NullPointerException` - if `key` is a `null` reference.
`IndexOutOfBoundsException` - if

- `keyOffset` is negative.
- `key` length is less than the sum of `keyOffset` and 16.

Methods

getAlgorithm

```
public String getAlgorithm()
```

Returns the name of the algorithm associated with this Square key.

Returns:
The string "Square".

getEncoded

```
public byte[] getEncoded()
```

(continued from last page)

Returns the key material of this Square key.

Each time this method is called, a new array is returned.

Returns:
The key material.

getFormat

`public String getFormat()`

Returns the name of the encoding format for this Square key.

Returns:
The string "RAW".

Package
com.imis.io

com.imis.io
Class BufferedInputStream

```
java.lang.Object
  |
  +-java.io.InputStream
      |
      +-com.imis.io.BufferedInputStream
```

All Implemented Interfaces:
Closeable

---

public abstract class **BufferedInputStream**
extends InputStream

Abstract base class that implements buffered read operations on a derived input stream.

Field Summary	
`protected`	`buf` The input stream buffer.
`protected`	`bufCount` The number of bytes read into the input stream buffer.
`protected`	`bufPos` The current position in the input stream buffer.
`protected`	`mark` The currently marked position in the stream, equal to the `pos` at the time the last `mark(int)` method was called or `-1` if the mark position was not set or exceeded the `readLimit` set by the `mark(int)` method.
`protected`	`markLimit` The maximum limit of bytes that can be read before the mark position becomes invalid.
`protected`	`needsSeek` The value indicating whether or not `fillBuffer()` method needs to seek by `pos` from beginning of the file.
`protected`	`pos` The input stream position.
`protected`	`size` The size, in bytes, of the available input stream data.

Constructor Summary	
`public`	`BufferedInputStream(int size)` Initializes a new instance of the `BufferedInputStream` class.
`public`	`BufferedInputStream(int size, int bufSize)` Initializes a new instance of the `BufferedInputStream` class with the specified read buffer size.

Method Summary

`int`	`available()` Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.
`void`	`close()` Closes this input stream and releases any system resources associated with the stream.
`abstract void`	`fillBuffer()` When overridden in a derived class, fills the data in the input stream buffer.
`void`	`mark(int readLimit)` Marks the current position in this input stream.
`boolean`	`markSupported()` Tests if this input stream supports the `mark` and `reset` methods.
`int`	`read()` Reads the next byte of data from the input stream.
`int`	`read(byte[] b)` Reads some number of bytes from the input stream and stores them into the buffer array `b`.
`int`	`read(byte[] b, int off, int len)` Reads up to `len` bytes of data from the input stream into an array of bytes.
`void`	`reset()` Repositions this stream to the position at the time the `mark(int)` method was last called on this input stream.
`long`	`skip(long n)` Skips over and discards `n` bytes of data from this input stream.

Methods inherited from class `java.io.InputStream`

`available, close, mark, markSupported, read, read, read, reset, skip`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.io.Closeable`

`close`

Fields

size

`protected long **size**`

The size, in bytes, of the available input stream data.

pos

`protected long **pos**`

(continued from last page)

The input stream position.

mark

`protected long mark`

The currently marked position in the stream, equal to the `pos` at the time the last `mark(int)` method was called or `-1` if the mark position was not set or exceeded the `readLimit` set by the `mark(int)` method.

markLimit

`protected int markLimit`

The maximum limit of bytes that can be read before the mark position becomes invalid.

needsSeek

`protected boolean needsSeek`

The value indicating whether or not `fillBuffer()` method needs to seek by `pos` from beginning of the file.

buf

`protected byte buf`

The input stream buffer.

bufPos

`protected int bufPos`

The current position in the input stream buffer.

bufCount

`protected int bufCount`

The number of bytes read into the input stream buffer.

Constructors

BufferedInputStream

`public BufferedInputStream(int size)`

Initializes a new instance of the `BufferedInputStream` class.

The size of the read buffer is by default 8192 bytes.

Parameters:
`size` - the size in bytes of the stream data.

BufferedInputStream

```
public BufferedInputStream(int size,
                           int bufSize)
```

Initializes a new instance of the `BufferedInputStream` class with the specified read buffer size.

(continued from last page)

Parameters:

`size` - the size in bytes of the stream data.

`bufSize` - the size of the read buffer.

Methods

fillBuffer

```
protected abstract void fillBuffer()
  throws IOException
```

When overridden in a derived class, fills the data in the input stream buffer. Overridden method should set `bufCount` to the number of bytes read (that is less or equal to buffer length) and `bufPos` to `0`.

Throws:

`IOException` - if an I/O error occurs.

available

```
public int available()
  throws IOException
```

Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream. The next caller might be the same thread or another thread.

Returns:

The number of bytes that can be read from this input stream without blocking.

Throws:

`IOException` - if an I/O error occurs.

close

```
public void close()
  throws IOException
```

Closes this input stream and releases any system resources associated with the stream.

Throws:

`IOException` - if an I/O error occurs.

mark

```
public void mark(int readLimit)
```

Marks the current position in this input stream. A subsequent call to the `reset` method repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

The `readLimit` arguments tells this input stream to allow that many bytes to be read before the mark position gets invalidated.

The general contract of mark is that, if the method `markSupported` returns `true`, the stream somehow remembers all the bytes read after the call to `mark` and stands ready to supply those same bytes again if and whenever the method `reset` is called. However, the stream is not required to remember any data at all if more than `readLimit` bytes are read from the stream before `reset` is called.

Parameters:

`readLimit` - the maximum limit of bytes that can be read before the mark position becomes invalid.

(continued from last page)

markSupported

```
public boolean markSupported()
```

Tests if this input stream supports the `mark` and `reset` methods.

Returns:
`true` since `BufferedInputStream` supports the `mark` and `reset` methods.

---

reset

```
public void reset()
  throws IOException
```

Repositions this stream to the position at the time the `mark(int)` method was last called on this input stream.

If mark position is relatively positioned before the start of the input stream buffer, the current read data is invalidated.

Throws:
`IOException` - if an I/O error occurs.

---

read

```
public int read()
  throws IOException
```

Reads the next byte of data from the input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned.

Returns:
The next byte of data, or `-1` if the end of the stream is reached.

Throws:
`IOException` - if an I/O error occurs.

---

read

```
public int read(byte[] b)
  throws IOException
```

Reads some number of bytes from the input stream and stores them into the buffer array `b`. The number of bytes actually read is returned as an integer.

Parameters:
`b` - the buffer into which the data is read.

Returns:
The total number of bytes read into the buffer, or `-1` is there is no more data because the end of the stream has been reached.

Throws:
`NullPointerException` - if `b` is a `null` reference.
`IOException` - if an I/O error occurs.

---

read

```
public int read(byte[] b,
         int off,
         int len)
  throws IOException
```

(continued from last page)

Reads up to `len` bytes of data from the input stream into an array of bytes. An attempt is made to read as many as `len` bytes, but a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.

Parameters:

`b` - the buffer into which the data is read.

`off` - the start offset in the array `b` at which the data is written.

`len` - the maximum number of bytes to read.

Returns:

The total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Throws:

`NullPointerException` - if `b` is a `null` reference.

`IndexOutOfBoundsException` - if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

`IOException` - if an I/O error occurs.

skip

```
public long skip(long n)
  throws IOException
```

Skips over and discards `n` bytes of data from this input stream. The `skip` method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly `0`. This may result from any of a number of conditions; reaching end of file before `n` bytes have been skipped is only one possibility. The actual number of bytes skipped is returned. If `n` is negative, no bytes are skipped.

Parameters:

`n` - the number of bytes to be skipped.

Returns:

The actual number of bytes skipped.

Throws:

`IOException` - if an I/O error occurs.

com.imis.io
Class BufferedOutputStream

```
java.lang.Object
  |
  +-java.io.OutputStream
      |
      +-com.imis.io.BufferedOutputStream
```

All Implemented Interfaces:
Flushable, Closeable

---

public abstract class **BufferedOutputStream**
extends OutputStream

Abstract base class that implements buffered write operations on a derived output stream.

Field Summary

`protected`	`buf` The output stream buffer.
`protected`	`bufCount` The number of bytes written to the output stream buffer.
`protected`	`isDirty` Indicates the data was written to this output stream.
`protected`	`pos` The output stream position.

Constructor Summary

`public`	`BufferedOutputStream()` Initializes a new instance of the `BufferedOutputStream` class.
`public`	`BufferedOutputStream(int bufSize)` Initializes a new instance of the `BufferedOutputStream` class with the specified write buffer size.

Method Summary

`void`	`close()` Closes this output stream and releases any system resources associated with this stream.
`void`	`flush()` Flushes this output stream and forces any buffered output bytes to be written out.
`abstract void`	`flushBuffer()` When overridden in a derived class, flushes the data in the output stream buffer.
`void`	`write(byte[] b)` Writes `b.length` bytes from the specified byte array to this output stream.

`void`	`write(byte[] b, int off, int len)` Writes `len` bytes from the specified byte array starting at offset `off` to this output stream.
`void`	`write(int b)` Writes the specified byte to this output stream.

Methods inherited from class `java.io.OutputStream`
`close, flush, write, write, write`

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.io.Closeable`
`close`

Methods inherited from interface `java.io.Flushable`
`flush`

Fields

pos

`protected long` **`pos`**

The output stream position.

isDirty

`protected boolean` **`isDirty`**

Indicates the data was written to this output stream.

buf

`protected byte` **`buf`**

The output stream buffer.

bufCount

`protected int` **`bufCount`**

The number of bytes written to the output stream buffer.

Constructors

BufferedOutputStream

`public` **`BufferedOutputStream`**`()`

(continued from last page)

Initializes a new instance of the `BufferedOutputStream` class.

The size of the write buffer is by default 8192 bytes.

BufferedOutputStream

```
public BufferedOutputStream(int bufSize)
```

Initializes a new instance of the `BufferedOutputStream` class with the specified write buffer size.

Parameters:
`bufSize` - the size of the write buffer.

Methods

flushBuffer

```
protected abstract void flushBuffer()
  throws IOException
```

When overridden in a derived class, flushes the data in the output stream buffer. Overridden method should set `bufCount` to `0` and leave `pos` as is.

Throws:
`IOException` - if an I/O error occurs.

write

```
public void write(int b)
  throws IOException
```

Writes the specified byte to this output stream. The general contract for `write` is that one byte is written to the output stream. The byte to be written is the eight low-order bits of the argument `b`. The 24 high-order bits of `b` are ignored.

Parameters:
`b` - the `byte`.

Throws:
`IOException` - if an I/O error occurs.

write

```
public void write(byte[] b)
  throws IOException
```

Writes `b.length` bytes from the specified byte array to this output stream. The general contract for `write(b)` is that it should have exactly the same effect as the call `write(b, 0, b.length)`.

Parameters:
`b` - the buffer with data.

Throws:
`NullPointerException` - if `b` is a `null` reference.
`IOException` - if an I/O error occurs.

(continued from last page)

write

```
public void write(byte[] b,
         int off,
         int len)
  throws IOException
```

Writes `len` bytes from the specified byte array starting at offset `off` to this output stream. The general contract for `write(b, off, len)` is that some of the bytes in the array `b` are written to the output stream in order; element `b[off]` is the first byte written and `b[off+len-1]` is the last byte written by this operation.

Parameters:

`b` - the buffer with data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

Throws:

`NullPointerException` - if `b` is a `null` reference.

`IndexOutOfBoundsException` - if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

`IOException` - if an I/O error occurs.

flush

```
public void flush()
  throws IOException
```

Flushes this output stream and forces any buffered output bytes to be written out. The general contract of `flush` is that calling it is an indication that, if any bytes previously written have been buffered by the implementation of the output stream, such bytes should immediately be written to their intended destination.

Throws:

`IOException` - if an I/O error occurs.

close

```
public void close()
  throws IOException
```

Closes this output stream and releases any system resources associated with this stream. The general contract of `close` is that it flushes the output stream and closes the output stream. A closed stream cannot perform output operations and cannot be reopened.

Throws:

`IOException` - if an I/O error occurs.

com.imis.io
Class LittleEndianDataInputStream

```
java.lang.Object
  |
  +-java.io.InputStream
      |
      +-java.io.FilterInputStream
          |
          +-com.imis.io.LittleEndianDataInputStream
```

All Implemented Interfaces:
DataInput, Closeable

public class **LittleEndianDataInputStream**
extends FilterInputStream
implements Closeable, DataInput

A class that implements `java.io.DataInput` interface for reading bytes from a specified stream and reconstructing from them data in any of the Java primitive types in little-endian order.

Fields inherited from class `java.io.FilterInputStream`
`in`

Constructor Summary	
`public`	`LittleEndianDataInputStream(InputStream in)` Initializes a new instance of the `LittleEndianDataInputStream` class.

Method Summary	
`boolean`	`readBoolean()` Reads one input byte and returns `true` if that byte is nonzero, `false` if that byte is zero.
`byte`	`readByte()` Reads and returns one input byte.
`char`	`readChar()` Reads an input char and returns the `char` value.
`double`	`readDouble()` Reads eight input bytes and returns a `double` value.
`float`	`readFloat()` Reads four input bytes and returns a `float` value.
`void`	`readFully(byte[] b)` Reads some bytes from an input stream and stores them into the buffer array `b`.
`void`	`readFully(byte[] b, int off, int len)` Reads `len` bytes from an input stream and stores them into the buffer array `b` starting at offset `off`.
`int`	`readInt()` Reads four input bytes and returns an `int` value.

`String`	`readLine()` Reads the next line of text from the input stream.
`long`	`readLong()` Reads eight input bytes and returns a `long` value.
`short`	`readShort()` Reads two input bytes and returns a `short` value.
`int`	`readUnsignedByte()` Reads one input byte, zero-extends it to type `int`, and returns the result, which is therefore in the range `0` through `255`.
`int`	`readUnsignedShort()` Reads two input bytes and returns an `int` value in the range `0` through `65535`.
`String`	`readUTF()` Reads in a string that has been encoded using a modified UTF-8 format.
`int`	`skipBytes(int n)` Makes an attempt to skip over `n` bytes of data from the input stream, discarding the skipped bytes.

Methods inherited from class `java.io.FilterInputStream`

`available, close, mark, markSupported, read, read, read, reset, skip`

Methods inherited from class `java.io.InputStream`

`available, close, mark, markSupported, read, read, read, reset, skip`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.io.Closeable`

`close`

Methods inherited from interface `java.io.DataInput`

`readBoolean, readByte, readChar, readDouble, readFloat, readFully, readFully, readInt, readLine, readLong, readShort, readUnsignedByte, readUnsignedShort, readUTF, skipBytes`

Constructors

LittleEndianDataInputStream

`public` **`LittleEndianDataInputStream`**`(InputStream in)`

Initializes a new instance of the `LittleEndianDataInputStream` class.

Parameters:

`in` - the underlying input stream.

Methods

readFully

```
public void readFully(byte[] b)
  throws IOException
```

Reads some bytes from an input stream and stores them into the buffer array `b`. The number of bytes read is equal to the length of `b`. If b.length is zero, then no bytes are read. Otherwise, the first byte read is stored into element b[0], the next one into b[1], and so on.

Parameters:
`b` - the buffer into which the data is read.

Throws:
`NullPointerException` - if `b` is a `null` reference.
`EOFException` - if this stream reaches the end before reading all the bytes.
`IOException` - if an I/O error occurs.

---

readFully

```
public void readFully(byte[] b,
         int off,
         int len)
  throws IOException
```

Reads `len` bytes from an input stream and stores them into the buffer array `b` starting at offset `off`.

Parameters:
`b` - the buffer into which the data is read.
`off` - an int specifying the offset into the data.
`len` - an int specifying the number of bytes to read.

Throws:
`NullPointerException` - if `b` is a `null` reference.
`IndexOutOfBoundsException` - if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.
`EOFException` - if this stream reaches the end before reading all the bytes.
`IOException` - if an I/O error occurs.

---

skipBytes

```
public int skipBytes(int n)
  throws IOException
```

Makes an attempt to skip over `n` bytes of data from the input stream, discarding the skipped bytes. However, it may skip over some smaller number of bytes, possibly zero.

Parameters:
`n` - the number of bytes to be skipped.

Returns:
The number of bytes actually skipped.

Throws:
`IOException` - if an I/O error occurs.

(continued from last page)

readBoolean

```
public boolean readBoolean()
  throws IOException
```

Reads one input byte and returns `true` if that byte is nonzero, `false` if that byte is zero. This method is suitable for reading the byte written by the `writeBoolean` method of interface `DataOutput`.

Returns:
The `boolean` value read.

Throws:
`EOFException` - if this stream reaches the end before reading all the bytes.
`IOException` - if an I/O error occurs.

readByte

```
public byte readByte()
  throws IOException
```

Reads and returns one input byte. The byte is treated as a signed value in the range `-128` through `127`, inclusive. This method is suitable for reading the byte written by the `writeByte` method of interface `DataOutput`.

Returns:
The 8-bit value read.

Throws:
`EOFException` - if this stream reaches the end before reading all the bytes.
`IOException` - if an I/O error occurs.

readUnsignedByte

```
public int readUnsignedByte()
  throws IOException
```

Reads one input byte, zero-extends it to type `int`, and returns the result, which is therefore in the range `0` through `255`. This method is suitable for reading the byte written by the `writeByte` method of interface `DataOutput` if the argument to `writeByte` was intended to be a value in the range `0` through `255`.

Returns:
The unsigned 8-bit value read.

Throws:
`EOFException` - if this stream reaches the end before reading all the bytes.
`IOException` - if an I/O error occurs.

readShort

```
public short readShort()
  throws IOException
```

Reads two input bytes and returns a `short` value. Let `a` be the first byte read and `b` be the second byte returned with `FilterInputStream.read()`. The value returned is:

```
(short)((b << 8) | a)
```

This method is suitable for reading the bytes written by the `writeShort` method of interface `DataOutput`.

Returns:
The 16-bit value read.

(continued from last page)

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readUnsignedShort

```
public int readUnsignedShort()
  throws IOException
```

Reads two input bytes and returns an `int` value in the range `0` through `65535`. Let `a` be the first byte read and `b` be the second byte returned with `FilterInputStream.read()`. The value returned is:

```
(b << 8) | a
```

This method is suitable for reading the bytes written by the `writeShort` method of interface `DataOutput` if the argument to `writeShort` was intended to be a value in the range `0` through `65535`.

Returns:

The unsigned 16-bit value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readChar

```
public char readChar()
  throws IOException
```

Reads an input char and returns the `char` value. A Unicode `char` is made up of two bytes. Let `a` be the first byte read and `b` be the second byte returned with `FilterInputStream.read()`. The value returned is:

```
(char)((b << 8) | a)
```

This method is suitable for reading bytes written by the `writeChar` method of interface `DataOutput`.

Returns:

The Unicode `char` read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readInt

```
public int readInt()
  throws IOException
```

Reads four input bytes and returns an `int` value. Let `a` be the first byte read, `b` be the second byte, `c` be the third byte, and `d` be the fourth byte returned with `FilterInputStream.read()`. The value returned is:

```
(d << 24) | (c << 16) | (b << 8) | a
```

This method is suitable for reading bytes written by the `writeInt` method of interface `DataOutput`.

Returns:

The `int` value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readLong

```
public long readLong()
  throws IOException
```

Reads eight input bytes and returns a `long` value. Let `a` be the first byte read, `b` be the second byte, `c` be the third byte, `d` be the fourth byte, `e` be the fifth byte, `f` be the sixth byte, `g` be the seventh byte, and `h` be the eighth byte returned with `FilterInputStream.read()`. The value returned is:

```
((long)a << 56) |
 (long)b << 48) |
 (long)c << 40) |
 (long)d << 32) |
 (long)e << 24) |
 (long)f << 16) |
 (long)g << 8) |
 (long)h)
```

This method is suitable for reading bytes written by the `writeLong` method of interface `DataOutput`.

Returns:
The `long` value read.

Throws:
`EOFException` - if this stream reaches the end before reading all the bytes.
`IOException` - if an I/O error occurs.

readFloat

```
public float readFloat()
  throws IOException
```

Reads four input bytes and returns a `float` value. It does this by first constructing an `int` value in exactly the manner of the `readInt` method, then converting this `int` value to a float in exactly the manner of the method `Float.intBitsToFloat`. This method is suitable for reading bytes written by the `writeFloat` method of interface `DataOutput`.

Returns:
The `float` value read.

Throws:
`EOFException` - if this stream reaches the end before reading all the bytes.
`IOException` - if an I/O error occurs.

readDouble

```
public double readDouble()
  throws IOException
```

Reads eight input bytes and returns a `double` value. It does this by first constructing a long value in exactly the manner of the `readLong` method, then converting this `long` value to a `double` in exactly the manner of the method `Double.longBitsToDouble`. This method is suitable for reading bytes written by the `writeDouble` method of interface `DataOutput`.

Returns:
The `double` value read.

Throws:
`EOFException` - if this stream reaches the end before reading all the bytes.
`IOException` - if an I/O error occurs.

readLine

```
public String readLine()
  throws IOException
```

Reads the next line of text from the input stream. It reads successive bytes, converting each byte separately into a character, until it encounters a line terminator or end of file; the characters read are then returned as a `String`. Note that because this method processes bytes, it does not support input of the full Unicode character set.

If end of file is encountered before even one byte can be read, then `null` is returned. Otherwise, each byte that is read is converted to type `char` by zero-extension. If the character `'\n'` is encountered, it is discarded and reading ceases. If the character `'\r'` is encountered, it is discarded and, if the following byte converts to the character `'\n'`, then that is discarded also; reading then ceases. If end of file is encountered before either of the characters `'\n'` and `'\r'` is encountered, reading ceases. Once reading has ceased, a `String` is returned that contains all the characters read and not discarded, taken in order. Note that every character in this string will have a value less than `?`, that is, `(char)256`.

Returns:

The next line of text from the input stream, or `null` if the end of file is encountered before a byte can be read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readUTF

```
public String readUTF()
  throws IOException
```

Reads in a string that has been encoded using a modified UTF-8 format. The general contract of `readUTF` is that it reads a representation of a Unicode character string encoded in Java modified UTF-8 format; this string of characters is then returned as a `String`.

The `writeUTF` method of interface `DataOutput` may be used to write data that is suitable for reading by this method.

Remarks: This method is not implemented!

Returns:

A Unicode string.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

com.imis.io
Class LittleEndianDataOutputStream

```
java.lang.Object
  |
  +-java.io.OutputStream
      |
      +-java.io.FilterOutputStream
          |
          +-com.imis.io.LittleEndianDataOutputStream
```

All Implemented Interfaces:
DataOutput, Flushable, Closeable

---

public class **LittleEndianDataOutputStream**
extends FilterOutputStream
implements Closeable, Flushable, DataOutput

A class that implements `java.io.DataOutput` interface for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a specified stream in little-endian order.

Field Summary

`protected`	`written` The size of the written data.

Fields inherited from class `java.io.FilterOutputStream`

`out`

Constructor Summary

`public`	`LittleEndianDataOutputStream(OutputStream out)` Initializes a new instance of the `LittleEndianDataOutputStream` class.

Method Summary

`int`	`size()` Returns the size of the written data.
`void`	`write(byte[] b)` Writes to the output stream all the bytes in array `b`.
`void`	`write(byte[] b, int off, int len)` Writes `len` bytes from array `b`, in order, to the output stream.
`void`	`write(int b)` Writes to the output stream the eight low-order bits of the argument `b`.
`void`	`writeBoolean(boolean v)` Writes a `boolean` value to this output stream.
`void`	`writeByte(int v)` Writes to the output stream the eight low- order bits of the argument `v`.

void	`writeBytes(String s)` Writes a string to the output stream.
void	`writeChar(int v)` Writes a `char` value, which is comprised of two bytes, to the output stream.
void	`writeChars(String s)` Writes every character in the string `s`, to the output stream, in order, two bytes per character.
void	`writeDouble(double v)` Writes a `double` value, which is comprised of eight bytes, to the output stream.
void	`writeFloat(float v)` Writes a `float` value, which is comprised of four bytes, to the output stream.
void	`writeInt(int v)` Writes an `int` value, which is comprised of four bytes, to the output stream.
void	`writeLong(long v)` Writes a `long` value, which is comprised of eight bytes, to the output stream.
void	`writeShort(int v)` Writes two bytes to the output stream to represent the value of the argument.
void	`writeUTF(String s)` Writes two bytes of length information to the output stream, followed by the Java modified UTF representation of every character in the string `s`.

Methods inherited from class `java.io.FilterOutputStream`
`close, flush, write, write, write`

Methods inherited from class `java.io.OutputStream`
`close, flush, write, write, write`

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.io.Closeable`
`close`

Methods inherited from interface `java.io.Flushable`
`flush`

Methods inherited from interface `java.io.DataOutput`
`write, write, write, writeBoolean, writeByte, writeBytes, writeChar, writeChars, writeDouble, writeFloat, writeInt, writeLong, writeShort, writeUTF`

Fields

(continued from last page)

written

```
protected int written
```

The size of the written data.

Constructors

LittleEndianDataOutputStream

```
public LittleEndianDataOutputStream(OutputStream out)
```

Initializes a new instance of the `LittleEndianDataOutputStream` class.

Parameters:
`out` - the underlying output stream.

Methods

size

```
public final int size()
```

Returns the size of the written data.

Returns:
The size of the written data.

---

write

```
public void write(int b)
  throws IOException
```

Writes to the output stream the eight low-order bits of the argument `b`. The 24 high-order bits of `b` are ignored.

Parameters:
`b` - the byte to be written.

Throws:
`IOException` - if an I/O error occurs.

---

write

```
public void write(byte[] b)
  throws IOException
```

Writes to the output stream all the bytes in array `b`. If `b.length` is zero, then no bytes are written. Otherwise, the byte `b[0]` is written first, then `b[1]`, and so on; the last byte written is `b[b.length-1]`.

Parameters:
`b` - the buffer that contains the data.

Throws:
`NullPointerException` - if `b` is a `null` reference.
`IOException` - if an I/O error occurs.

(continued from last page)

write

```
public void write(byte[] b,
         int off,
         int len)
  throws IOException
```

Writes `len` bytes from array `b`, in order, to the output stream. If `len` is zero, then no bytes are written. Otherwise, the byte `b[off]` is written first, then `b[off+1]`, and so on; the last byte written is `b[off+len-1]`.

Parameters:

`b` - the byte array containing the data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

Throws:

`NullPointerException` - if `b` is a `null` reference.

`IndexOutOfBoundsException` - if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

`IOException` - if an I/O error occurs.

---

writeBoolean

```
public final void writeBoolean(boolean v)
  throws IOException
```

Writes a `boolean` value to this output stream. If the argument `v` is true, the value `(byte)1` is written; if `v` is false, the value `(byte)0` is written. The byte written by this method may be read by the `readBoolean` method of interface `DataInput`, which will then return a `boolean` equal to `v`.

Parameters:

`v` - the `boolean` value to be written.

Throws:

`IOException` - if an I/O error occurs.

---

writeByte

```
public final void writeByte(int v)
  throws IOException
```

Writes to the output stream the eight low- order bits of the argument `v`. The 24 high-order bits of v are ignored. (This means that `writeByte` does exactly the same thing as write for an integer argument.) The byte written by this method may be read by the `readByte` method of interface `DataInput`, which will then return a `byte` equal to `(byte)v`.

Parameters:

`v` - the `byte` value to be written.

Throws:

`IOException` - if an I/O error occurs.

---

writeShort

```
public final void writeShort(int v)
  throws IOException
```

(continued from last page)

Writes two bytes to the output stream to represent the value of the argument. The byte values to be written, in the order shown, are:

```
(byte)(0xff & v) (byte)(0xff & (v >> 8))
```

The bytes written by this method may be read by the `readShort` method of interface `DataInput`, which will then return a `short` equal to `(short)v`.

Parameters:
`v` - the `short` value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeChar

```
public final void writeChar(int v)
  throws IOException
```

Writes a `char` value, which is comprised of two bytes, to the output stream. The byte values to be written, in the order shown, are:

```
(byte)(0xff & v) (byte)(0xff & (v >> 8))
```

The bytes written by this method may be read by the `readChar` method of interface `DataInput`, which will then return a `char` equal to `(char)v`.

Parameters:
`v` - the `char` value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeInt

```
public final void writeInt(int v)
  throws IOException
```

Writes an `int` value, which is comprised of four bytes, to the output stream. The byte values to be written, in the order shown, are:

```
(byte)(0xff & v) (byte)(0xff & (v >> 8))
 (byte)(0xff & (v >> 16))
 (byte)(0xff & (v >> 24))
```

The bytes written by this method may be read by the `readInt` method of interface `DataInput`, which will then return an `int` equal to `v`.

Parameters:
`v` - the `int` value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeLong

```
public final void writeLong(long v)
  throws IOException
```

(continued from last page)

Writes a `long` value, which is comprised of eight bytes, to the output stream. The byte values to be written, in the order shown, are:

```
(byte)(0xff & v) (byte)(0xff & (v >> 8))
 (byte)(0xff & (v >> 16))
 (byte)(0xff & (v >> 24))
 (byte)(0xff & (v >> 32))
 (byte)(0xff & (v >> 40))
 (byte)(0xff & (v >> 48))
 (byte)(0xff & (v >> 56))
```

The bytes written by this method may be read by the `readLong` method of interface `DataInput`, which will then return a `long` equal to `v`.

Parameters:
`v` - the `long` value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeFloat

```
public final void writeFloat(float v)
  throws IOException
```

Writes a `float` value, which is comprised of four bytes, to the output stream. It does this as if it first converts this `float` value to an `int` in exactly the manner of the `Float.floatToIntBits` method and then writes the `int` value in exactly the manner of the `writeInt` method. The bytes written by this method may be read by the `readFloat` method of interface `DataInput`, which will then return a `float` equal to `v`.

Parameters:
`v` - the `float` value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeDouble

```
public final void writeDouble(double v)
  throws IOException
```

Writes a `double` value, which is comprised of eight bytes, to the output stream. It does this as if it first converts this `double` value to a `long` in exactly the manner of the `Double.doubleToLongBits` method and then writes the long value in exactly the manner of the `writeLong` method. The bytes written by this method may be read by the `readDouble` method of interface `DataInput`, which will then return a `double` equal to `v`.

Parameters:
`v` - the `double` value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeBytes

```
public final void writeBytes(String s)
  throws IOException
```

(continued from last page)

Writes a string to the output stream. For every character in the string `s`, taken in order, one byte is written to the output stream. If `s` is `null`, a `NullPointerException` is thrown.

If `s.length` is zero, then no bytes are written. Otherwise, the character `s[0]` is written first, then `s[1]`, and so on; the last character written is `s[s.length-1]`. For each character, one byte is written, the low-order byte, in exactly the manner of the `writeByte` method. The high-order eight bits of each character in the string are ignored.

Parameters:
`s` - the string of bytes to be written.

Throws:
`IOException` - if an I/O error occurs.

writeChars

```
public final void writeChars(String s)
  throws IOException
```

Writes every character in the string `s`, to the output stream, in order, two bytes per character. If `s` is `null`, a `NullPointerException` is thrown. If `s.length` is zero, then no characters are written. Otherwise, the character `s[0]` is written first, then `s[1]`, and so on; the last character written is `s[s.length-1]`. For each character, two bytes are actually written, high-order byte first, in exactly the manner of the `writeChar` method.

Parameters:
`s` - the string value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeUTF

```
public void writeUTF(String s)
  throws IOException
```

Writes two bytes of length information to the output stream, followed by the Java modified UTF representation of every character in the string `s`. If `s` is `null`, a `NullPointerException` is thrown. Each character in the string `s` is converted to a group of one, two, or three bytes, depending on the value of the character.

The bytes written by this method may be read by the `readUTF` method of interface `DataInput`, which will then return a `String` equal to `s`.

Remarks: This method is not implemented!

Parameters:
`s` - the string value to be written.

Throws:
`IOException` - if an I/O error occurs.

com.imis.io
Class Streams

```
java.lang.Object
  |
  +-com.imis.io.Streams
```

public class **Streams**
extends Object

Provides methods for working with streams.

Constructor Summary

public	`Streams()`

Method Summary

static long	`copy(InputStream in, OutputStream out)` Copies the contents from the input stream to the output stream.
static boolean	`equals(InputStream in1, InputStream in2)` Compare the contents of two streams to determine if they are equal or not.
static byte[]	`toByteArray(InputStream in)` Copies the contents of the input stream to an array of bytes.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

Streams

```
public Streams()
```

Methods

copy

```
public static long copy(InputStream in,
          OutputStream out)
  throws IOException
```

Copies the contents from the input stream to the output stream.

This method does not close or flush either stream.

Parameters:

(continued from last page)

`in` - the input stream.
`out` - the output stream.

Returns:
The total number of bytes copied from the input stream.

Throws:
`NullPointerException` - if `in` or `out` is a `null` reference.
`IOException` - if an I/O error occurs.

equals

```
public static boolean equals(InputStream in1,
         InputStream in2)
  throws IOException
```

Compare the contents of two streams to determine if they are equal or not.

This method buffers the input internally using `java.io.BufferedInputStream` if they are not already buffered.

Parameters:
`in1` - the first stream.
`in2` - the second stream.

Returns:
`true` if the content of the streams are equal; otherwise `false`.

Throws:
`IOException` - if an I/O error occurs.

toByteArray

```
public static byte[] toByteArray(InputStream in)
  throws IOException
```

Copies the contents of the input stream to an array of bytes.

Parameters:
`in` - the input stream.

Returns:
An array of bytes containing the content of the input stream.

Throws:
`NullPointerException` - if `in` is a `null` reference.
`IOException` - if an I/O error occurs.

Package
com.imis.net

com.imis.net
Class InetServices

```
java.lang.Object
   |
   +-com.imis.net.InetServices
```

public final class **InetServices**
extends Object

Provides the means to get the port number of a well-known service.

Table of known and supported `os.home` system property values and the location of the `services` file in that operation system.

os.home property value	**services file location**
AIX	/etc/services
Digital Unix	/etc/services
FreeBSD	/etc/services
HP UX	/etc/services
Irix	/etc/services
Linux	/etc/services
Mac OS	/etc/services
Mac OS X	/etc/services
MPE/iX	/etc/services
Netware 4.11	not supported
OS/2	not supported
OS/390	/etc/services
Solaris	/etc/services
SunOS	/etc/services
Windows 2000	%SystemRoot%\system32\drivers\etc\services
Windows 2003	%SystemRoot%\system32\drivers\etc\services
Windows 7	%SystemRoot%\system32\drivers\etc\services
Windows 95	%SystemRoot%\system32\drivers\etc\services
Windows 98	%SystemRoot%\system32\drivers\etc\services
Windows NT	%SystemRoot%\system32\drivers\etc\services
Windows Vista	%SystemRoot%\system32\drivers\etc\services
Windows XP	%SystemRoot%\system32\drivers\etc\services

Author:
Robert Petek

Method Summary

`static int`	`getPort(String service, String protocol)` Gets the port number of the well-known service found in the `services` file.
`static Integer`	`tryGetPort(String service, String protocol)` Gets the port number of the well-known service found in the `services` file.

Methods inherited from class `java.lang.Object`

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
wait
```

Methods

getPort

```
public static int getPort(String service,
        String protocol)
```

Gets the port number of the well-known service found in the `services` file.

Parameters:
`service` - the name of a well-known service found in the `services` file.
`protocol` - either `tcp` or `udp`, depending on the well-known service desired.

Returns:
A port number for a well-known service.

Throws:
`NullPointerException` - if `service` or `protocol` is a `null` reference.
`IllegalArgumentException` - if `protocol` value is other then `tcp` or `udp`.
`NoSuchElementException` - if `service` is not one of the well-known service names.
`UnsupportedOperationException` - if

- Access to the `os.name` or `java.library.path` system property not allowed.
- System property `os.name` or `java.library.path` does not exist.
- Unsupported platform for service to port translation.
- Services file does not exists.
- Error reading services file.

tryGetPort

```
public static Integer tryGetPort(String service,
        String protocol)
```

Gets the port number of the well-known service found in the `services` file.

Parameters:
`service` - the name of a well-known service found in the `services` file.
`protocol` - either `tcp` or `udp`, depending on the well-known service desired.

Returns:
A port number for a well-known service or `null` if `service` is not one of the well-known service names.

Throws:
`NullPointerException` - if `service` or `protocol` is a `null` reference.
`IllegalArgumentException` - if `protocol` value is other then `tcp` or `udp`.

Package
com.imis.security

com.imis.security
Class MessageDigestMD2

```
java.lang.Object
   |
   +-java.security.MessageDigestSpi
      |
      +-java.security.MessageDigest
         |
         +-com.imis.security.MessageDigestMD2
```

public class **MessageDigestMD2**
extends java.security.MessageDigest

An implementation of the MD2 message digest algorithm.

Note: MD2 is not widely used. Unless it is needed for compatibility with existing systems, it is not recommended for use in new applications.

References:

1. The MD2 Message-Digest Algorithm, B. Kaliski.
2. The RFC ERRATA PAGE under section RFC 1319.

Test vectors:

Input	**Message digest**
""	8350E5A3E24C153DF2275C9F80692773
"a"	32EC01EC4A6DAC72C0AB96FB34C0B5D1
"abc"	DA853B0D3F88D99B30283A69E6DED6BB
"message digest"	AB4F496BFB2A530B219FF33031FE06B0
"abcdefghijklmnopqrstuvwxyz"	4E8DDFF3650292AB5A4108C3AA47940B
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"	DA33DEF2A42DF13975352846C30338CD
"12345678901234567890123456789012345678901234567890123456789012345678901234567890"	D5976F79D83D3A0DC9806C3C66F3EFD8

Constructor Summary

public	`MessageDigestMD2()` Initializes a new instance of the `MessageDigestMD2` class.
public	`MessageDigestMD2(MessageDigestMD2 md2)` Initializes a new instance of the `MessageDigestMD2` class with an instance of the `MessageDigestMD2` class.

Method Summary

`byte[]`	`engineDigest()` Completes the hash computation by performing final operations and resets the engine.
`int`	`engineGetDigestLength()` Returns the digest length in bytes.

void	`engineReset()` Resets the digest for further use.
void	`engineUpdate(byte input)` Updates the digest using the specified byte.
void	`engineUpdate(byte[] input, int offset, int len)` Updates the digest using the specified array of bytes, starting at the specified offset.
void	`processBlock(byte[] input)` Processes the block.
void	`processCheckSum(byte[] input)` Processes the check sum.

Methods inherited from class `java.security.MessageDigest`

`clone, digest, digest, digest, getAlgorithm, getDigestLength, getInstance, getInstance, getInstance, getProvider, isEqual, reset, toString, update, update, update, update`

Methods inherited from class `java.security.MessageDigestSpi`

`clone, engineDigest, engineDigest, engineGetDigestLength, engineReset, engineUpdate, engineUpdate, engineUpdate`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

MessageDigestMD2

`public` **`MessageDigestMD2`**`()`

Initializes a new instance of the `MessageDigestMD2` class.

MessageDigestMD2

`public` **`MessageDigestMD2`**`(MessageDigestMD2 md2)`

Initializes a new instance of the `MessageDigestMD2` class with an instance of the `MessageDigestMD2` class.

This is a copy constructor. We are using copy constructors in place of the `Object.clone()` interface as this interface is not supported by J2ME.

Parameters:
`md2` - a `MessageDigestMD2` instance.

Methods

processCheckSum

`protected void` **`processCheckSum`**`(byte[] input)`

(continued from last page)

Processes the check sum.

Parameters:
`input` - the array of bytes.

processBlock

```
protected void processBlock(byte[] input)
```

Processes the block.

Parameters:
`input` - the array of bytes.

engineUpdate

```
protected void engineUpdate(byte input)
```

Updates the digest using the specified byte.

Parameters:
`input` - the byte to use for the update.

engineUpdate

```
protected void engineUpdate(byte[] input,
         int offset,
         int len)
```

Updates the digest using the specified array of bytes, starting at the specified offset.

Parameters:
`input` - the array of bytes to use for the update.
`offset` - the offset to start from in the array of bytes.
`len` - the number of bytes to use, starting at `offset`.

engineDigest

```
protected byte[] engineDigest()
```

Completes the hash computation by performing final operations and resets the engine.

Returns:
The array of bytes for the resulting hash value.

engineReset

```
protected void engineReset()
```

Resets the digest for further use.

engineGetDigestLength

```
protected int engineGetDigestLength()
```

Returns the digest length in bytes.

Returns:

(continued from last page)

The digest length in bytes.

com.imis.security
Class MessageDigestMD4

```
java.lang.Object
  |
  +-java.security.MessageDigestSpi
      |
      +-java.security.MessageDigest
          |
          +-com.imis.security.MessageDigestMD4
```

public class **MessageDigestMD4**
extends java.security.MessageDigest

Implementation of MD4 as RFC 1320 by R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc.

Note: This algorithm is only included for backwards compatibility with legacy applications. It is not secure or to be used for anything new.

Test vectors:

Input	**Message digest**
""	31D6CFE0D16AE931B73C59D7E0C089C0
"a"	BDE52CB31DE33E46245E05FBDBD6FB24
"abc"	A448017AAF21D8525FC10AE87AA6729D
"message digest"	D9130A8164549FE818874806E1C7014B
"abcdefghijklmnopqrstuvwxyz"	D79E1C308AA5BBCDEEA8ED63DF412DA9
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"	043F8582F241DB351CE627E153E7F0E4
"12345678901234567890123456789012345678901234567890123456789012345678901234567890"	E33B4DDC9C38F2199C3E7B164FCC0536

Constructor Summary

`public`	`MessageDigestMD4()` Initializes a new instance of the `MessageDigestMD4` class.
`protected`	`MessageDigestMD4(MessageDigestMD4 md4)` Initializes a new instance of the `MessageDigestMD4` class with an instance of the `MessageDigestMD4` class.

Method Summary

`byte[]`	`engineDigest()` Completes the hash computation by performing final operations and resets the engine.
`int`	`engineGetDigestLength()` Returns the digest length in bytes.
`void`	`engineReset()` Resets the digest for further use.
`void`	`engineUpdate(byte input)` Updates the digest using the specified byte.

void	`engineUpdate(byte[] input, int offset, int len)` Updates the digest using the specified array of bytes, starting at the specified offset.

Methods inherited from class `java.security.MessageDigest`
`clone, digest, digest, digest, getAlgorithm, getDigestLength, getInstance, getInstance, getInstance, getProvider, isEqual, reset, toString, update, update, update, update`

Methods inherited from class `java.security.MessageDigestSpi`
`clone, engineDigest, engineDigest, engineGetDigestLength, engineReset, engineUpdate, engineUpdate, engineUpdate`

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

MessageDigestMD4

```
public MessageDigestMD4()
```

Initializes a new instance of the `MessageDigestMD4` class.

MessageDigestMD4

```
protected MessageDigestMD4(MessageDigestMD4 md4)
```

Initializes a new instance of the `MessageDigestMD4` class with an instance of the `MessageDigestMD4` class.

This is a copy constructor. We are using copy constructors in place of the `Object.clone()` interface as this interface is not supported by J2ME.

Parameters:
`md4` - a `MessageDigestMD4` instance.

Methods

engineUpdate

```
protected void engineUpdate(byte input)
```

Updates the digest using the specified byte.

Parameters:
`input` - the byte to use for the update.

engineUpdate

```
protected void engineUpdate(byte[] input,
         int offset,
         int len)
```

(continued from last page)

Updates the digest using the specified array of bytes, starting at the specified offset.

Parameters:

`input` - the array of bytes to use for the update.
`offset` - the offset to start from in the array of bytes.
`len` - the number of bytes to use, starting at `offset`.

engineDigest

`protected byte[] **engineDigest**()`

Completes the hash computation by performing final operations and resets the engine.

Returns:

The array of bytes for the resulting hash value.

engineReset

`protected void **engineReset**()`

Resets the digest for further use.

engineGetDigestLength

`protected int **engineGetDigestLength**()`

Returns the digest length in bytes.

Returns:

The digest length in bytes.

com.imis.security
Class MessageDigestSHA0

```
java.lang.Object
  |
  +-java.security.MessageDigestSpi
      |
      +-java.security.MessageDigest
          |
          +-com.imis.security.MessageDigestSHA0
```

public class **MessageDigestSHA0**
extends java.security.MessageDigest

Implementation of Secure Hash Standard, FIPS PUB 180, 1993, now often referred to as SHA-0.

This is implementation of Secure Hash Standard is based on RSA library.

Copyright (c) J.S.A.Kapp 1994 - 1996. (port to java by Jure Puhek (Imaging Systems)

RSAEURO - RSA Library compatible with RSAREF(tm) 2.0.

All functions prototypes are the Same as for RSAREF(tm). To aid compatibility the source and the files follow the same naming conventions that RSAREF(tm) uses. This should aid direct importing to your applications.

This library is legal everywhere outside the US. And should NOT be imported to the US and used there.

All Trademarks Acknowledged.

Test Vectors:

Input	**Message digest**
""	F96CEA198AD1DD5617AC084A3D92C6107708C0EF
"a"	37F297772FAE4CB1BA39B6CF9CF0381180BD62F2
"abc"	0164B8A914CD2A5E74C4F7FF082C4D97F1EDF880
"message digest"	C1B0F222D150EBB9AA36A40CAFDC8BCBED830B14
"abcdefghijklmnopqrstuvwxyz"	B40CE07A430CFD3C033039B9FE9AFEC95DC1BDCD
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopq rstuvwxyz0123456789"	79E966F7A3A990DF33E40E3D7F8F18D2CAEBADFA
"12345678901234567890123456789012345678901234567890 123456789012345678901234567890"	4AA29D14D171522ECE47BEE8957E35A41F3E9CFF

Constructor Summary

`public`	`MessageDigestSHA0()` Initializes a new instance of the `MessageDigestSHA0` class.
`public`	`MessageDigestSHA0(MessageDigestSHA0 sha0)` Initializes a new instance of the `MessageDigestSHA0` class.

Method Summary

`byte[]`	`engineDigest()` Completes the hash computation by performing final operations and resets the engine.

int	`engineGetDigestLength()` Returns the digest length in bytes.
void	`engineReset()` Resets the digest for further use.
void	`engineUpdate(byte input)` Updates the digest using the specified byte.
void	`engineUpdate(byte[] input, int offset, int len)` Updates the digest using the specified array of bytes, starting at the specified offset.

Methods inherited from class `java.security.MessageDigest`

`clone, digest, digest, digest, getAlgorithm, getDigestLength, getInstance, getInstance, getInstance, getProvider, isEqual, reset, toString, update, update, update, update`

Methods inherited from class `java.security.MessageDigestSpi`

`clone, engineDigest, engineDigest, engineGetDigestLength, engineReset, engineUpdate, engineUpdate, engineUpdate`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

MessageDigestSHA0

`public` **`MessageDigestSHA0`**`()`

Initializes a new instance of the `MessageDigestSHA0` class.

MessageDigestSHA0

`public` **`MessageDigestSHA0`**`(MessageDigestSHA0 sha0)`

Initializes a new instance of the `MessageDigestSHA0` class.

This is a copy constructor. We are using copy constructors in place of the `Object.clone()` interface as this interface is not supported by J2ME.

Parameters:
`sha0` - a `MessageDigestSHA0` instance.

Methods

engineUpdate

`protected void` **`engineUpdate`**`(byte input)`

Updates the digest using the specified byte.

(continued from last page)

Parameters:
`input` - the byte to use for the update.

engineUpdate

```
protected void engineUpdate(byte[] input,
         int offset,
         int len)
```

Updates the digest using the specified array of bytes, starting at the specified offset.

Parameters:
`input` - the array of bytes to use for the update.
`offset` - the offset to start from in the array of bytes.
`len` - the number of bytes to use, starting at `offset`.

engineDigest

```
protected byte[] engineDigest()
```

Completes the hash computation by performing final operations and resets the engine.

Returns:
The array of bytes for the resulting hash value.

engineReset

```
protected void engineReset()
```

Resets the digest for further use.

engineGetDigestLength

```
protected int engineGetDigestLength()
```

Returns the digest length in bytes.

Returns:
The digest length in bytes.

com.imis.security
Class MessageDigestSHA1

```
java.lang.Object
   |
   +-java.security.MessageDigestSpi
      |
      +-java.security.MessageDigest
         |
         +-com.imis.security.MessageDigestSHA1
```

public class **MessageDigestSHA1**
extends java.security.MessageDigest

Implementation of the SHA-1 message digest algorithm.

This is implementation of SHA-1 is based on `MessageDigestSHA0`}, the only difference between the two is an extra bitwise rotation before the execution of the 80 steps of processing message digest. **References:**

1. NIST FIPS PUB 180-1, "Secure Hash Standard", U.S. Department of Commerce, May 1993.
   http://www.itl.nist.gov/div897/pubs/fip180-1.htm

Test Vectors:

Input	**Message digest**
""	DA39A3EE5E6B4B0D3255BFEF95601890AFD80709
"a"	86F7E437FAA5A7FCE15D1DDCB9EAEAEA377667B8
"abc"	A9993E364706816ABA3E25717850C26C9CD0D89D
"message digest"	C12252CEDA8BE8994D5FA0290A47231C1D16AAE3
"abcdefghijklmnopqrstuvwxyz"	32D10C7B8CF96570CA04CE37F2A19D84240D3A89
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"	761C457BF73B14D27E9E9265C46F4B4DDA11F940
"12345678901234567890123456789012345678901234567890123456789012345678901234567890"	50ABF5706A150990A08B2C5EA40FA0E585554732

Constructor Summary

`public`	`MessageDigestSHA1()` Initializes a new instance of the `MessageDigestSHA1` class.
`public`	`MessageDigestSHA1(MessageDigestSHA1 sha1)` Initializes a new instance of the `MessageDigestSHA1` class.

Method Summary

`byte[]`	`engineDigest()` Completes the hash computation by performing final operations and resets the engine.
`int`	`engineGetDigestLength()` Returns the digest length in bytes.
`void`	`engineReset()` Resets the digest for further use.

`void`	`engineUpdate(byte input)` Updates the digest using the specified byte.
`void`	`engineUpdate(byte[] input, int offset, int len)` Updates the digest using the specified array of bytes, starting at the specified offset.

Methods inherited from class `java.security.MessageDigest`
`clone, digest, digest, digest, getAlgorithm, getDigestLength, getInstance, getInstance, getInstance, getProvider, isEqual, reset, toString, update, update, update, update`

Methods inherited from class `java.security.MessageDigestSpi`
`clone, engineDigest, engineDigest, engineGetDigestLength, engineReset, engineUpdate, engineUpdate, engineUpdate`

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

MessageDigestSHA1

`public MessageDigestSHA1()`

Initializes a new instance of the `MessageDigestSHA1` class.

MessageDigestSHA1

`public MessageDigestSHA1(MessageDigestSHA1 sha1)`

Initializes a new instance of the `MessageDigestSHA1` class.

This is a copy constructor. We are using copy constructors in place of the `Object.clone()` interface as this interface is not supported by J2ME.

Parameters:
`sha1` - a `MessageDigestSHA1` instance.

Methods

engineUpdate

`protected void engineUpdate(byte input)`

Updates the digest using the specified byte.

Parameters:
`input` - the byte to use for the update.

(continued from last page)

engineUpdate

```
protected void engineUpdate(byte[] input,
         int offset,
         int len)
```

Updates the digest using the specified array of bytes, starting at the specified offset.

Parameters:

`input` - the array of bytes to use for the update.
`offset` - the offset to start from in the array of bytes.
`len` - the number of bytes to use, starting at `offset`.

engineDigest

```
protected byte[] engineDigest()
```

Completes the hash computation by performing final operations and resets the engine.

Returns:

The array of bytes for the resulting hash value.

engineReset

```
protected void engineReset()
```

Resets the digest for further use.

engineGetDigestLength

```
protected int engineGetDigestLength()
```

Returns the digest length in bytes.

Returns:

The digest length in bytes.

com.imis.security
Class MessageDigestSHA256

```
java.lang.Object
   |
   +-java.security.MessageDigestSpi
         |
         +-java.security.MessageDigest
               |
               +-com.imis.security.MessageDigestSHA256
```

public class **MessageDigestSHA256**
extends java.security.MessageDigest

Implementation of the SHA-256 message digest algorithm.

References:

1. Federal Information, Processing Standards Publication 180-2, 2002 August 1, SECURE HASH STANDARD
   http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

Test Vectors:

Input	Message digest
""	E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855
"a"	CA978112CA1BBDCAFAC231B39A23DC4DA786EFF8147C4E72B9807785AFEE48BB
"abc"	BA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410FF61F20015AD
"message digest"	F7846F55CF23E14EEBEAB5B4E1550CAD5B509E3348FBC4EFA3A1413D393CB650
"abcdefghijklmnopqrstuvwxyz"	71C480DF93D6AE2F1EFAD1447C66C9525E316218CF51FC8D9ED832F2DAF18B73
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"	DB4BFCBD4DA0CD85A60C3C37D3FBD8805C77F15FC6B1FDFE614EE0A7C8FDB4C0
"12345678901234567890123456789012345678901234567890123456789012345678901234567890"	F371BC4A311F2B009EEF952DD83CA80E2B60026C8E935592D0F9C308453C813E

Constructor Summary

`public`	`MessageDigestSHA256()` Initializes a new instance of the `MessageDigestSHA256` class.
`public`	`MessageDigestSHA256(MessageDigestSHA256 sha256)` Initializes a new instance of the `MessageDigestSHA256` class.

Method Summary

`byte[]`	`engineDigest()` Completes the hash computation by performing final operations and resets the engine.
`int`	`engineGetDigestLength()` Returns the digest length in bytes.

void	`engineReset()` Resets the digest for further use.
void	`engineUpdate(byte input)` Updates the digest using the specified byte.
void	`engineUpdate(byte[] input, int offset, int len)` Updates the digest using the specified array of bytes, starting at the specified offset.

Methods inherited from class `java.security.MessageDigest`
`clone, digest, digest, digest, getAlgorithm, getDigestLength, getInstance, getInstance, getInstance, getProvider, isEqual, reset, toString, update, update, update, update`

Methods inherited from class `java.security.MessageDigestSpi`
`clone, engineDigest, engineDigest, engineGetDigestLength, engineReset, engineUpdate, engineUpdate, engineUpdate`

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

MessageDigestSHA256

`public` **`MessageDigestSHA256`**`()`

Initializes a new instance of the `MessageDigestSHA256` class.

MessageDigestSHA256

`public` **`MessageDigestSHA256`**`(MessageDigestSHA256 sha256)`

Initializes a new instance of the `MessageDigestSHA256` class.

This is a copy constructor. We are using copy constructors in place of the `Object.clone()` interface as this interface is not supported by J2ME.

Parameters:
`sha256` - a `MessageDigestSHA256` instance.

Methods

engineUpdate

`protected void` **`engineUpdate`**`(byte input)`

Updates the digest using the specified byte.

Parameters:
`input` - the byte to use for the update.

engineUpdate

```
protected void engineUpdate(byte[] input,
         int offset,
         int len)
```

Updates the digest using the specified array of bytes, starting at the specified offset.

Parameters:

`input` - the array of bytes to use for the update.
`offset` - the offset to start from in the array of bytes.
`len` - the number of bytes to use, starting at `offset`.

engineDigest

```
protected byte[] engineDigest()
```

Completes the hash computation by performing final operations and resets the engine.

Returns:

The array of bytes for the resulting hash value.

engineReset

```
protected void engineReset()
```

Resets the digest for further use.

engineGetDigestLength

```
protected int engineGetDigestLength()
```

Returns the digest length in bytes.

Returns:

The digest length in bytes.

com.imis.security
Class MessageDigestTiger

```
java.lang.Object
   |
   +-java.security.MessageDigestSpi
      |
      +-java.security.MessageDigest
         |
         +-com.imis.security.MessageDigestTiger
```

public class **MessageDigestTiger**
extends java.security.MessageDigest

Implementation of the 192-bit Tiger algorithm, by Ross Anderson and Eli Biham, based on the sample C code published by Eli Biham found on http://www.cs.technion.ac.il/~biham/Reports/Tiger/.

Test Vectors:

Input	**Message digest**
""	3293AC630C13F0245F92BBB1766E16167A4E58492DDE73F3
"a"	77BEFBEF2E7EF8AB2EC8F93BF587A7FC613E247F5F247809
"abc"	2AAB1484E8C158F2BFB8C5FF41B57A525129131C957B5F93
"message digest"	D981F8CB78201A950DCF3048751E441C517FCA1AA55A29F6
"abcdefghijklmnopqrstuvwxyz"	1714A472EEE57D30040412BFCC55032A0B11602FF37BEEE9
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"	8DCEA680A17583EE502BA38A3C368651890FFBCCDC49A8CC
"12345678901234567890123456789012345678901234567890123456789012345678901234567890"	1C14795529FD9F207A958F84C52F11E887FA0CABDFD91BFD

Author:
Robert Petek

Constructor Summary

`public`	`MessageDigestTiger()` Initializes a new instance of the `MessageDigestTiger` class.
`public`	`MessageDigestTiger(MessageDigestTiger tiger)` Initializes a new instance of the `MessageDigestTiger` class.

Method Summary

`byte[]`	`engineDigest()` Completes the hash computation by performing final operations and resets the engine.
`int`	`engineGetDigestLength()` Returns the digest length in bytes.
`void`	`engineReset()` Resets the digest for further use.

`void`	`engineUpdate(byte input)` Updates the digest using the specified byte.
`void`	`engineUpdate(byte[] input, int offset, int len)` Updates the digest using the specified array of bytes, starting at the specified offset.

Methods inherited from class `java.security.MessageDigest`

`clone, digest, digest, digest, getAlgorithm, getDigestLength, getInstance, getInstance, getInstance, getProvider, isEqual, reset, toString, update, update, update, update`

Methods inherited from class `java.security.MessageDigestSpi`

`clone, engineDigest, engineDigest, engineGetDigestLength, engineReset, engineUpdate, engineUpdate, engineUpdate`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

MessageDigestTiger

`public MessageDigestTiger()`

Initializes a new instance of the `MessageDigestTiger` class.

MessageDigestTiger

`public MessageDigestTiger(MessageDigestTiger tiger)`

Initializes a new instance of the `MessageDigestTiger` class.

This is a copy constructor. We are using copy constructors in place of the `Object.clone()` interface as this interface is not supported by J2ME.

Parameters:
`tiger` - a `MessageDigestTiger` instance.

Methods

engineUpdate

`protected void engineUpdate(byte input)`

Updates the digest using the specified byte.

Parameters:
`input` - the byte to use for the update.

(continued from last page)

engineUpdate

```
protected void engineUpdate(byte[] input,
         int offset,
         int len)
```

Updates the digest using the specified array of bytes, starting at the specified offset.

Parameters:

`input` - the array of bytes to use for the update.
`offset` - the offset to start from in the array of bytes.
`len` - the number of bytes to use, starting at `offset`.

engineDigest

```
protected byte[] engineDigest()
```

Completes the hash computation by performing final operations and resets the engine.

Returns:

The array of bytes for the resulting hash value.

engineReset

```
protected void engineReset()
```

Resets the digest for further use.

engineGetDigestLength

```
protected int engineGetDigestLength()
```

Returns the digest length in bytes.

Returns:

The digest length in bytes.

com.imis.security
Class Provider

```
java.lang.Object
  |
  +-java.util.Dictionary
      |
      +-java.util.Hashtable
          |
          +-java.util.Properties
              |
              +-java.security.Provider
                  |
                  +-com.imis.security.Provider
```

All Implemented Interfaces:
Serializable, Cloneable, Map

public final class **Provider**
extends java.security.Provider

The Imaging Systems JCE Crypto Provider, a Java Security API provider that provides implementations of following cryptographic algorithms:

Symmetric Ciphers:

- **Square**: The 128-bit key, 128-bit block cipher algorithm developed by Joan Daemen, Lars Knudsen and Vincent Rijmen.

Message Digests:

- **MD2**: The MD2 message digest algorithm as defined in RFC 1319.
- **MD4**: The MD2 message digest algorithm as defined in RFC 1320.
- **SHA-0**: The Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180
- **SHA-1**: The Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-1.
- **SHA-256**: The Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-2.
- **Tiger**: The Tiger message digest algorithm, designed by Ross Anderson and Eli Biham

Author:
Robert Petek

Field Summary

`public static final`	`NAME` Name of the Imaging Systems JCE Crypto Provider. Value: **`ImagingSystemsJCE`**

Fields inherited from class `java.util.Properties`

`defaults`

Constructor Summary

`public`	`Provider()` Initializes a new instance of the `Provider` class.

Methods inherited from class `java.security.Provider` clear, elements, entrySet, get, getInfo, getName, getProperty, getService, getServices, getVersion, keys, keySet, load, put, putAll, putService, remove, removeService, toString, values

Methods inherited from class `java.util.Properties`

getProperty, getProperty, list, list, load, loadFromXML, propertyNames, save, setProperty, store, storeToXML, storeToXML

Methods inherited from class `java.util.Hashtable`

clear, clone, contains, containsKey, containsValue, elements, entrySet, equals, get, hashCode, isEmpty, keys, keySet, put, putAll, rehash, remove, size, toString, values

Methods inherited from class `java.util.Dictionary`

elements, get, isEmpty, keys, put, remove, size

Methods inherited from class `java.lang.Object`

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface `java.util.Map`

clear, containsKey, containsValue, entrySet, equals, get, hashCode, isEmpty, keySet, put, putAll, remove, size, values

Fields

NAME

`public static final java.lang.String` **`NAME`**

Name of the Imaging Systems JCE Crypto Provider.
Constant value: **`ImagingSystemsJCE`**

Constructors

Provider

`public` **`Provider`**`()`

Initializes a new instance of the `Provider` class.

Package
com.imis.security.auth.srp

com.imis.security.auth.srp
Class SRPAuthenticator

```
java.lang.Object
   |
   +-com.imis.security.auth.srp.SRPAuthenticator
```

public class **SRPAuthenticator**
extends Object

Authenticator object for the SRP-6a Secure Remote Password protocol implementation.

Constructor Summary

public	`SRPAuthenticator(java.security.MessageDigest md, byte[] prime, byte[] generator)` Initializes a new instance of the `SRPAuthenticator` class.

Method Summary

`SRPClientContext`	`createClientContext(byte[] userName, byte[] password)` Creates a `SRPClientContext` object.
`SRPServerContext`	`createServerContext(byte[] userName, byte[] salt, byte[] verifier)` Creates a `SRPServerContext` object.
`SRPVerifier`	`generateVerifier(byte[] password, int maxSalt)` Generates a `SRPVerifier` object.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

SRPAuthenticator

```
public SRPAuthenticator(java.security.MessageDigest md,
                        byte[] prime,
                        byte[] generator)
```

Initializes a new instance of the `SRPAuthenticator` class.

Parameters:
`md` - the `java.security.MessageDigest`.
`prime` - the large safe prime.
`generator` - the generator (modulo `prime`).

Throws:
`NullPointerException` - if `md`, `prime` or `generator` is a `null` reference.

Methods

(continued from last page)

createClientContext

```
public SRPClientContext createClientContext(byte[] userName,
          byte[] password)
```

Creates a SRPClientContext object.

Parameters:

`userName` - the user name.
`password` - the password.

Returns:

The `SRPClientContext` object.

Throws:

`NullPointerException` - if `userName` or `password` is a `null` reference.

createServerContext

```
public SRPServerContext createServerContext(byte[] userName,
          byte[] salt,
          byte[] verifier)
  throws Exception
```

Creates a SRPServerContext object.

Parameters:

`userName` - the user name.
`salt` - the user's salt.
`verifier` - the password verifier.

Returns:

The `SRPServerContext` object.

Throws:

`NullPointerException` - if `userName`, `salt` or `verifier` is a `null` reference.

generateVerifier

```
public SRPVerifier generateVerifier(byte[] password,
          int maxSalt)
  throws SRPAuthException
```

Generates a SRPVerifier object.

Parameters:

`password` - the password.
`maxSalt` - the maximum possible value of salt in bits.

Returns:

The `SRPVerifier` object.

Throws:

`NullPointerException` - if `password` is a `null` reference.
`IllegalArgumentException` - if `maxSalt` is less than `8`.
`SRPAuthException` - if salt not set.

com.imis.security.auth.srp
Class SRPAuthException

```
java.lang.Object
  |
  +-java.lang.Throwable
      |
      +-java.lang.Exception
          |
          +-com.imis.security.auth.srp.SRPAuthException
```

All Implemented Interfaces:
Serializable

public class **SRPAuthException**
extends Exception

Represents errors that occur in SRP-6a Secure Remote Password protocol implementation.

Constructor Summary

public	`SRPAuthException(String message)` Initializes a new instance of the `SRPAuthException` class with the specified detail message describing the exception.
public	`SRPAuthException(String message, Object arg0)` Initializes a new instance of the `SRPAuthException` class with the specified formatted detail message describing the exception.
public	`SRPAuthException(String message, Object[] args)` Initializes a new instance of the `SRPAuthException` class with the specified formatted detail message describing the exception.
public	`SRPAuthException(String message, Throwable cause)` Initializes a new instance of the `SRPAuthException` class with the detail message describing the exception and a reference to the throwable that is the cause of this exception.
public	`SRPAuthException(String message, Throwable cause, Object arg0)` Initializes a new instance of the `SRPAuthException` class with the specified formatted detail message describing the exception and a reference to the throwable that is the cause of this exception.
public	`SRPAuthException(String message, Throwable cause, Object[] args)` Initializes a new instance of the `SRPAuthException` class with the specified formatted detail message describing the exception and a reference to the throwable that is the cause of this exception.

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

SRPAuthException

```
public SRPAuthException(String message)
```

Initializes a new instance of the `SRPAuthException` class with the specified detail message describing the exception.

Parameters:

`message` - the detail message.

SRPAuthException

```
public SRPAuthException(String message,
                        Object arg0)
```

Initializes a new instance of the `SRPAuthException` class with the specified formatted detail message describing the exception.

Parameters:

`message` - the detail message containing one format item.

`arg0` - an object to format.

SRPAuthException

```
public SRPAuthException(String message,
                        Object[] args)
```

Initializes a new instance of the `SRPAuthException` class with the specified formatted detail message describing the exception.

Parameters:

`message` - the detail message containing zero or more format items.

`args` - an object array containing zero or more objects to format.

SRPAuthException

```
public SRPAuthException(String message,
                        Throwable cause)
```

Initializes a new instance of the `SRPAuthException` class with the detail message describing the exception and a reference to the throwable that is the cause of this exception.

Parameters:

`message` - the detail message.

`cause` - the throwable that is the cause of the current exception, or a `null` reference if no cause is specified.

SRPAuthException

```
public SRPAuthException(String message,
                        Throwable cause,
                        Object arg0)
```

Initializes a new instance of the `SRPAuthException` class with the specified formatted detail message describing the exception and a reference to the throwable that is the cause of this exception.

Parameters:

(continued from last page)

`message` - the detail message describing the exception and containing one format item.
`cause` - the throwable that is the cause of the current exception, or a `null` reference if no cause is specified.
`arg0` - an object to format.

SRPAuthException

```
public SRPAuthException(String message,
                        Throwable cause,
                        Object[] args)
```

Initializes a new instance of the `SRPAuthException` class with the specified formatted detail message describing the exception and a reference to the throwable that is the cause of this exception.

Parameters:

`message` - the detail message containing zero or more format items.
`cause` - the throwable that is the cause of the current exception, or a `null` reference if no cause is specified.
`args` - an object array containing zero or more objects to format.

com.imis.security.auth.srp
Class SRPClientContext

```
java.lang.Object
  |
  +-com.imis.security.auth.srp.SRPClientContext
```

public class **SRPClientContext**
extends Object

Client context for the SRP-6a Secure Remote Password protocol implementation.

Method Summary

`byte[]`	`generateEvidence()` Generates client side evidence and calculates the session key.
`byte[]`	`getPublicKey()` Gets client public key.
`byte[]`	`getSessionKey()` Gets the session key.
`void`	`setSalt(byte[] salt)` Sets the user's salt.
`void`	`setServerPublicKey(byte[] key)` Sets and validates the server public key.
`boolean`	`validate(byte[] serverEvidence)` Validates the server evidence.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

getPublicKey

```
public byte[] getPublicKey()
```

Gets client public key.

Returns:
The client public key.

getSessionKey

```
public byte[] getSessionKey()
  throws SRPAuthException
```

(continued from last page)

Gets the session key.

Returns:
The session key.

Throws:
`SRPAuthException` - if evidence not generated.

setSalt

```
public void setSalt(byte[] salt)
```

Sets the user's salt.

Parameters:
`salt` - the user's salt.

Throws:
`NullPointerException` - if `salt` is a `null` reference.

setServerPublicKey

```
public void setServerPublicKey(byte[] key)
  throws SRPAuthException
```

Sets and validates the server public key.

Parameters:
`key` - the server public key.

Throws:
`NullPointerException` - if `key` is a `null` reference.
`SRPAuthException` - if server public key is invalid.

generateEvidence

```
public byte[] generateEvidence()
  throws SRPAuthException
```

Generates client side evidence and calculates the session key.

Returns:
The client evidence.

Throws:
`SRPAuthException` - if

- Server public key not set.
- User's salt not set.

validate

```
public boolean validate(byte[] serverEvidence)
  throws SRPAuthException
```

Validates the server evidence.

Parameters:

(continued from last page)

`serverEvidence` - the server evidence.

Returns:

`true` if the server and client evidence match; otherwise `false`.

Throws:

`NullPointerException` - if `serverEvidence` is a `null` reference.
`SRPAuthException` - if evidence not generated.

com.imis.security.auth.srp
Class SRPServerContext

```
java.lang.Object
  |
  +-com.imis.security.auth.srp.SRPServerContext
```

public class **SRPServerContext**
extends Object

Server context for the SRP-6a Secure Remote Password protocol implementation.

Method Summary

`void`	`generateEvidence()` Generates server side evidence and calculates the session key.
`byte[]`	`getPublicKey()` Gets the server public key.
`byte[]`	`getSalt()` Gets the user's salt.
`byte[]`	`getSessionKey()` Gets the session key.
`void`	`setClientPublicKey(byte[] key)` Sets and validates the client public key.
`byte[]`	`validate(byte[] clientEvidence)` Validates the client evidence.

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

getPublicKey

```
public byte[] getPublicKey()
```

Gets the server public key.

Returns:
The server public key.

getSalt

```
public byte[] getSalt()
```

Gets the user's salt.

(continued from last page)

Returns:
The user's salt.

getSessionKey

```
public byte[] getSessionKey()
  throws SRPAuthException
```

Gets the session key.

Returns:
The session key.

Throws:
`SRPAuthException` - if evidence not generated.

setClientPublicKey

```
public void setClientPublicKey(byte[] key)
  throws SRPAuthException
```

Sets and validates the client public key.

The public key `A` must match `A % N != 0`.

Parameters:
`key` - the client public key.

Throws:
`NullPointerException` - if `key` is a `null` reference.
`SRPAuthException` - if client public key is invalid.

generateEvidence

```
public void generateEvidence()
  throws SRPAuthException
```

Generates server side evidence and calculates the session key.

Throws:
`SRPAuthException` - if client public key not set.

validate

```
public byte[] validate(byte[] clientEvidence)
  throws SRPAuthException
```

Validates the client evidence.

Parameters:
`clientEvidence` - the client evidence.

Returns:
The server evidence.

Throws:
`NullPointerException` - if `clientEvidence` is a `null` reference.

(continued from last page)

`SRPAuthException` - if

- Evidence not generated.
- Server and client evidence do not match.

com.imis.security.auth.srp
Class SRPUtils

```
java.lang.Object
  |
  +-com.imis.security.auth.srp.SRPUtils
```

public final class **SRPUtils**
extends Object

Provides methods used in SRP-6a Secure Remote Password protocol implementation.

Constructor Summary

`public`	`SRPUtils()`

Method Summary

`static byte[]`	`createRandomBytes(int count)` Creates an array with the specified number of random bytes.
`static java.math.BigInteger`	`createRandomNumber(java.math.BigInteger n)` Creates a random number modulo `n` that satisfies the condition: `1 < random < n`.

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

SRPUtils

`public SRPUtils()`

Methods

createRandomNumber

`public static java.math.BigInteger createRandomNumber(java.math.BigInteger n)`

Creates a random number modulo `n` that satisfies the condition: `1 < random < n`.

Parameters:
`n` - the modulo.

Returns:
A random number.

Throws:

(continued from last page)

`NullPointerException` - if `n` is a `null` reference.
`IllegalArgumentException` - if `n` less than or equal to `2`.

createRandomBytes

`public static byte[] **createRandomBytes**(int count)`

Creates an array with the specified number of random bytes.

Parameters:
`count` - the number of requested random bytes.

Returns:
An array with `count` random bytes.

Throws:
`IllegalArgumentException` - if `count` is negative.

com.imis.security.auth.srp
Class SRPVerifier

```
java.lang.Object
  |
  +-com.imis.security.auth.srp.SRPVerifier
```

public class **SRPVerifier**
extends Object

Verifier object for the SRP-6a Secure Remote Password protocol implementation.

Defines storage for password verifier and corresponding salt value.

Method Summary

`byte[]`	`getSalt()` Gets the user's salt.
`byte[]`	`getVerifier()` Gets the password verifier.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

getVerifier

`public byte[] getVerifier()`

Gets the password verifier.

Returns:
The password verifier.

getSalt

`public byte[] getSalt()`

Gets the user's salt.

Returns:
The user's salt.

Package
com.imis.text

com.imis.text
Class DateFormat

```
java.lang.Object
  |
  +-com.imis.text.DateFormat
```

public class **DateFormat**
extends Object

Thread safe date and time formatter.

Field Summary

`public static final`	`PATTERN_DATE` The date format pattern ("dd.MM.yyyy"). Value: **`dd.MM.yyyy`**
`public static final`	`PATTERN_DATE_TIME` The date and time format pattern ("dd.MM.yyyy HH:mm:ss"). Value: **`dd.MM.yyyy HH:mm:ss`**
`public static final`	`PATTERN_TIME` The date format pattern ("HH:mm:ss"). Value: **`HH:mm:ss`**

Constructor Summary

`public`	`DateFormat()`

Method Summary

`static String`	`format(Date date)` Formats `Date` into string representation of its date and time value using default time zone and date time pattern.
`static String`	`format(Date date, String pattern)` Formats `Date` into string representation of its date and time value using default time zone and the specified pattern.
`static String`	`format(Date date, TimeZone zone)` Formats `Date` into string representation of its date and time value using the specified time zone and default date time pattern.
`static String`	`format(Date date, TimeZone zone, String pattern)` Formats `Date` into string representation of its date and time value using the specified time zone and pattern.
`static Date`	`parse(String source)` Parses the given string to produce a `Date` using default time zone and date time pattern.
`static Date`	`parse(String source, String pattern)` Parses the given string to produce a `Date` using default time zone and the specified pattern.

`static Date`	`parse(String source, TimeZone zone)` Parses the given string to produce a `Date` using the specified time zone and default date time pattern.
`static Date`	`parse(String source, TimeZone zone, String pattern)` Parses the given string to produce a `Date` using the specified time zone and pattern.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

PATTERN_DATE

`public static final java.lang.String` **`PATTERN_DATE`**

The date format pattern ("dd.MM.yyyy").
Constant value: **`dd.MM.yyyy`**

PATTERN_TIME

`public static final java.lang.String` **`PATTERN_TIME`**

The date format pattern ("HH:mm:ss").
Constant value: **`HH:mm:ss`**

PATTERN_DATE_TIME

`public static final java.lang.String` **`PATTERN_DATE_TIME`**

The date and time format pattern ("dd.MM.yyyy HH:mm:ss").
Constant value: **`dd.MM.yyyy HH:mm:ss`**

Constructors

DateFormat

`public` **`DateFormat`**`()`

Methods

format

`public static String` **`format`**`(Date date)`

Formats `Date` into string representation of its date and time value using default time zone and date time pattern.

Parameters:
`date` - the `Date` to be formatted into a date and time string.

Returns:
The formatted date and time string.

(continued from last page)

Throws:
`NullPointerException` - if `date` is `null` reference.

See Also:
`java.text.SimpleDateFormat`

format

```
public static String format(Date date,
         String pattern)
```

Formats `Date` into string representation of its date and time value using default time zone and the specified pattern.

Parameters:
`date` - the `Date` to be formatted into a date and time string.
`pattern` - a pattern string describing this date and time format.

Returns:
The formatted date and time string.

Throws:
`NullPointerException` - if `date` or `pattern` is `null` reference.
`IllegalArgumentException` - if `pattern` is invalid.

See Also:
`java.text.SimpleDateFormat`

format

```
public static String format(Date date,
         TimeZone zone)
```

Formats `Date` into string representation of its date and time value using the specified time zone and default date time pattern.

Parameters:
`date` - the `Date` to be formatted into a date and time string.
`zone` - the time zone associated with the calendar of `DateFormat`.

Returns:
The formatted date and time string.

Throws:
`NullPointerException` - if `date` or `zone` is `null` reference.

See Also:
`java.text.SimpleDateFormat`

format

```
public static String format(Date date,
         TimeZone zone,
         String pattern)
```

Formats `Date` into string representation of its date and time value using the specified time zone and pattern.

Parameters:
`date` - the `Date` to be formatted into a date and time string.

(continued from last page)

`zone` - the time zone associated with the calendar of `DateFormat`.
`pattern` - a pattern string describing this date and time format.

Returns:
The formatted date and time string.

Throws:
`NullPointerException` - if `date`, `zone` or `pattern` is `null` reference.
`IllegalArgumentException` - if `pattern` is invalid.

See Also:
`java.text.SimpleDateFormat`

---

parse

```
public static Date parse(String source)
  throws java.text.ParseException
```

Parses the given string to produce a `Date` using default time zone and date time pattern.

Parameters:
`source` - a string to be parsed.

Returns:
A `Date` instance.

Throws:
`NullPointerException` - if `source` is `null` reference.
`ParseException` - if `source` cannot be parsed.

---

parse

```
public static Date parse(String source,
         String pattern)
  throws java.text.ParseException
```

Parses the given string to produce a `Date` using default time zone and the specified pattern.

Parameters:
`source` - a string to be parsed.
`pattern` - a pattern string describing this date and time format.

Returns:
A `Date` instance.

Throws:
`NullPointerException` - if `source`, `zone` or `pattern` is `null` reference.
`ParseException` - if `source` cannot be parsed.

---

parse

```
public static Date parse(String source,
         TimeZone zone)
  throws java.text.ParseException
```

Parses the given string to produce a `Date` using the specified time zone and default date time pattern.

Parameters:
`source` - a string to be parsed.

(continued from last page)

`zone` - the time zone associated with the calendar of `DateFormat`.

Returns:

A `Date` instance.

Throws:

`NullPointerException` - if `source` or `zone` is `null` reference.
`ParseException` - if `source` cannot be parsed.

---

parse

```
public static Date parse(String source,
         TimeZone zone,
         String pattern)
  throws java.text.ParseException
```

Parses the given string to produce a `Date` using the specified time zone and pattern.

Parameters:

`source` - a string to be parsed.
`zone` - the time zone associated with the calendar of `DateFormat`.
`pattern` - a pattern string describing this date and time format.

Returns:

A `Date` instance.

Throws:

`NullPointerException` - if `source`, `zone` or `pattern` is `null` reference.
`ParseException` - if `source` cannot be parsed.

Package
com.imis.util

com.imis.util
Class Arrays

```
java.lang.Object
  |
  +-com.imis.util.Arrays
```

public class **Arrays**
extends Object

Provides methods for working with array of bytes.

Constructor Summary

`public`	`Arrays()`

Method Summary

`static int`	`compare(byte[] a, byte[] b)` Compares two specified array of bytes and returns a value indicating whether one is less than, equal to, or greater than the other.
`static byte[]`	`concat(byte[] a, byte[] b)` Concatenates two arrays of bytes.
`static byte[]`	`copyOf(byte[] a, int length)` Copies the specified array of bytes, truncating or padding with zeros (if necessary) so the copy has the specified length.
`static byte[]`	`copyOfRange(byte[] a, int offset, int length)` Copies the specified range of the specified array of bytes into a new array of bytes.
`static byte[]`	`xor(byte[] a, byte[] b)` Performs XOR operation of two arrays of bytes.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

Arrays

`public **Arrays**()`

Methods

(continued from last page)

compare

```
public static int compare(byte[] a,
         byte[] b)
```

Compares two specified array of bytes and returns a value indicating whether one is less than, equal to, or greater than the other.

Parameters:

`a` - the first array of bytes.

`b` - the second array of bytes.

Returns:

A signed number indicating the relative values of this two arrays.

Value	Condition
Less than zero	`a` is less than `b`.
Zero	`a` equals `b`.
Greater than zero	`a` is greater than `b`.

concat

```
public static byte[] concat(byte[] a,
         byte[] b)
```

Concatenates two arrays of bytes.

Parameters:

`a` - the first array of bytes.

`b` - the second array of bytes.

Returns:

The concatenated array of bytes.

Throws:

`NullPointerException` - if `a` or `b` is a `null` reference.

copyOf

```
public static byte[] copyOf(byte[] a,
         int length)
```

Copies the specified array of bytes, truncating or padding with zeros (if necessary) so the copy has the specified length.

Parameters:

`a` - the array of bytes to be copied.

`length` - the length of the copy to be returned.

Returns:

The copy of `a`, truncated or padded with zeros to obtain the specified length.

Throws:

`NullPointerException` - if `a` is a `null` reference.

`IndexOutOfBoundsException` - if `length` is negative.

(continued from last page)

copyOfRange

```
public static byte[] copyOfRange(byte[] a,
         int offset,
         int length)
```

Copies the specified range of the specified array of bytes into a new array of bytes.

Parameters:

`a` - the array of bytes to be copied at the specified offset.
`offset` - the byte offset at which to begin copying array of bytes.
`length` - the length of the copy to be returned.

Returns:

The copy of `a` at the specified offset, truncated or padded with zeros to obtain the specified length.

Throws:

`NullPointerException` - if `a` is a `null` reference.
`IndexOutOfBoundsException` - if `offset` is negative, or `length` is negative, or `offset` greater than the length of `a`.

xor

```
public static byte[] xor(byte[] a,
         byte[] b)
```

Performs XOR operation of two arrays of bytes.

Parameters:

`a` - the first array of bytes.
`b` - the second array of bytes.

Returns:

The array of bytes with XOR-ed values.

Throws:

`NullPointerException` - if `a` or `b` is a `null` reference.
`IllegalArgumentException` - if `a` and `b` are not the same length.

com.imis.util
Class BitVector32

```
java.lang.Object
  |
  +-com.imis.util.BitVector32
```

public final class **BitVector32**
extends Object

Provides a simple class that stores boolean values and small integers in 32 bits of memory.

Nested Class Summary

`class`	`BitVector32.Section` BitVector32.Section

Constructor Summary

`public`	`BitVector32(int data)` Initializes a new instance of the `BitVector32` class with the specified internal data.
`public`	`BitVector32(BitVector32 value)` Initializes a new instance of the `BitVector32` class with the data represented in an existing `BitVector32` class.

Method Summary

`static int`	`createMask()` Creates the first mask in a series of masks that can be used to retrieve individual bits in a `BitVector32` that is set up as bit flags.
`static int`	`createMask(int previous)` Creates an additional mask following the specified mask in a series of masks that can be used to retrieve individual bits in a `BitVector32` that is set up as bit flags.
`static BitVector32.Section`	`createSection(short maxValue)` Creates the first `BitVector32.Section` in a series of sections that contain small integers.
`static BitVector32.Section`	`createSection(short maxValue, BitVector32.Section previous)` Creates a new `BitVector32.Section` following the specified `BitVector32.Section` in a series of sections that contain integers.
`boolean`	`equals(Object obj)` Determines whether the specified object is equal to the `BitVector32`.
`boolean`	`getBit(int bit)` Gets the value of the specified bit flag or section.
`int`	`getData()` Gets the value of the `BitVector32` as an `int` integer.
`int`	`getValue(BitVector32.Section section)` Gets the value stored in the specified `BitVector32.Section`.

`int`	`hashCode()` Serves as a hash function for the `BitVector32`.
`void`	`setBit(int bit, boolean value)` Sets the value of the specified bit flag or section.
`void`	`setValue(BitVector32.Section section, int value)` Sets the value stored in the specified `BitVector32.Section`.
`String`	`toString()` Returns a string that represents the current `BitVector32`.
`static String`	`toString(BitVector32 value)` Returns a string that represents the specified `BitVector32`.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

BitVector32

`public BitVector32(int data)`

Initializes a new instance of the `BitVector32` class with the specified internal data.

Parameters:
`data` - an integer representing the data of the new `BitVector32`.

BitVector32

`public BitVector32(BitVector32 value)`

Initializes a new instance of the `BitVector32` class with the data represented in an existing `BitVector32` class.

Parameters:
`value` - a `BitVector32` structure that contains the data to copy.

Throws:
`NullPointerException` - if `value` is `null` reference.

Methods

getBit

`public boolean getBit(int bit)`

Gets the value of the specified bit flag or section.

Parameters:
`bit` - a mask that indicates the bit to get

Returns:
`true` if the specified bit flag is on 1; otherwise, `false`.

setBit

```
public void setBit(int bit,
         boolean value)
```

Sets the value of the specified bit flag or section.

Parameters:
`bit` - a mask that indicates the bit to set.
`value` - `true` to set the specified bit flag to 1; otherwise, `false`.

getValue

```
public int getValue(BitVector32.Section section)
```

Gets the value stored in the specified `BitVector32.Section`.

Parameters:
`section` - a `BitVector32.Section` that contains the value to get or set.

Returns:
The value stored in the specified `BitVector32.Section`.

setValue

```
public void setValue(BitVector32.Section section,
         int value)
```

Sets the value stored in the specified `BitVector32.Section`.

Parameters:
`section` - a `BitVector32.Section` that contains the value to get or set.
`value` - the `int` value to be stored in the specified `BitVector32.Section`.

getData

```
public int getData()
```

Gets the value of the `BitVector32` as an `int` integer.

Returns:
The value of the `BitVector32` as an integer.

createMask

```
public static int createMask()
```

Creates the first mask in a series of masks that can be used to retrieve individual bits in a `BitVector32` that is set up as bit flags.

Returns:
A mask that isolates the first bit flag in the `BitVector32`.

createMask

```
public static int createMask(int previous)
  throws IllegalStateException
```

(continued from last page)

Creates an additional mask following the specified mask in a series of masks that can be used to retrieve individual bits in a `BitVector32` that is set up as bit flags.

Parameters:

`previous` - - the mask that indicates the previous bit flag.

Returns:

A mask that isolates the bit flag following the one that `previous` points to in `BitVector32`.

Throws:

`IllegalStateException` - if `previous` indicates the last bit flag in the `BitVector32`.

---

createSection

```
public static BitVector32.Section createSection(short maxValue)
  throws IllegalArgumentException
```

Creates the first `BitVector32.Section` in a series of sections that contain small integers.

Parameters:

`maxValue` - a 16-bit signed integer that specifies the maximum value for the new `BitVector32.Section`.

Returns:

A `BitVector32.Section` that can hold a number from zero to `maxValue`.

Throws:

`IllegalArgumentException` - if `maxValue` is less than 1.

---

createSection

```
public static BitVector32.Section createSection(short maxValue,
         BitVector32.Section previous)
  throws IllegalArgumentException
```

Creates a new `BitVector32.Section` following the specified `BitVector32.Section` in a series of sections that contain integers.

Parameters:

`maxValue` - a 16-bit signed integer that specifies the maximum value for the new `BitVector32.Section`.
`previous` - the previous `BitVector32.Section` in the `BitVector32`.

Returns:

A `BitVector32.Section` that can hold a number from zero to `maxValue`.

Throws:

`IllegalArgumentException` - if `maxValue` is less than 1.
`IllegalStateException` - if

- `previous` includes the final bit in the `BitVector32`.
- `maxValue` is greater than the highest value that can be represented by the number of bits after `previous`.

---

equals

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

Determines whether the specified object is equal to the `BitVector32`.

Parameters:

(continued from last page)

`obj` - the object to compare with the current `BitVector32`.

Returns:
`true` if the specified object is equal to the `BitVector32`; otherwise, `false`.

hashCode

`public int **hashCode**()`

Serves as a hash function for the `BitVector32`.

The hash code of a `BitVector32` is based on the value of `getData()`. Two instances of `BitVector32` with the same value for `Data` will also generate the same hash code.

Returns:
A hash code for the `BitVector32`.

toString

`public String **toString**()`

Returns a string that represents the current `BitVector32`.

Returns:
A string that represents the current `BitVector32`.

toString

`public static String **toString**(BitVector32 value)`

Returns a string that represents the specified `BitVector32`.

Parameters:
`value` - the `BitVector32` to represent as string.

Returns:
A string that represents the specified `BitVector32`.

com.imis.util
Class BitVector32.Section

```
java.lang.Object
  |
  +-com.imis.util.BitVector32.Section
```

public static final class **BitVector32.Section**
extends Object

Represents an section of the vector that can contain a integer number.

Method Summary

`boolean`	`equals(BitVector32.Section obj)` Determines whether the specified `BitVector32.Section` object is the same as the current `BitVector32.Section` object.
`boolean`	`equals(Object obj)` Determines whether the specified object is the same as the current `BitVector32.Section` object.
`int`	`GetHashCode()` Serves as a hash function for the current `BitVector32.Section`, suitable for hashing algorithms and data structures, such as a hash table.
`short`	`getMask()` Gets a mask that isolates this section within the `BitVector32`.
`short`	`getOffset()` Gets the offset of this section from the start of the `BitVector32`.
`String`	`toString()` Returns a string that represents the current `BitVector32.Section`.
`static String`	`toString(BitVector32.Section value)` Returns a string that represents the specified `BitVector32.Section`.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

getMask

`public short getMask()`

Gets a mask that isolates this section within the `BitVector32`.

Returns:
A mask that isolates this section within the `BitVector32`.

getOffset

`public short getOffset()`

Gets the offset of this section from the start of the `BitVector32`.

Returns:
The offset of this section from the start of the `BitVector32`.

equals

`public boolean equals(BitVector32.Section obj)`

Determines whether the specified `BitVector32.Section` object is the same as the current `BitVector32.Section` object.

Parameters:
`obj` - the `BitVector32.Section` object to compare with the current `BitVector32.Section` object.

Returns:
`true` if the `obj` parameter is the same as the current `BitVector32.Section` object; otherwise `false`.

equals

`public boolean equals(Object obj)`

Determines whether the specified object is the same as the current `BitVector32.Section` object.

Returns:
`true` if the specified object is the same as the current `BitVector32.Section` object; otherwise, `false`.

GetHashCode

`public int GetHashCode()`

Serves as a hash function for the current `BitVector32.Section`, suitable for hashing algorithms and data structures, such as a hash table.

This method generates the same hash code for two objects that are equal according to the `equals(BitVector32.Section)` method.

Returns:
A hash code for the current `BitVector32.Section`.

toString

`public String toString()`

Returns a string that represents the current `BitVector32.Section`.

Returns:
A string that represents the current `BitVector32.Section`.

toString

`public static String toString(BitVector32.Section value)`

Returns a string that represents the specified `BitVector32.Section`.

(continued from last page)

Parameters:

`value` - the `BitVector32.Section` to represent as string.

Returns:

A string that represents the specified `BitVector32.Section`.

com.imis.util
Class BitVector64

```
java.lang.Object
  |
  +-com.imis.util.BitVector64
```

public final class **BitVector64**
extends Object

Provides a simple structure that stores boolean values and integers in 64 bits of memory.

Nested Class Summary

class	`BitVector64.Section` BitVector64.Section

Constructor Summary

public	`BitVector64(int data)` Initializes a new instance of the `BitVector64` class containing the data represented in an integer.
public	`BitVector64(long data)` Initializes a new instance of the `BitVector64` class containing the data represented in a long integer.
public	`BitVector64(BitVector32 value)` Initializes a new instance of the `BitVector64` class containing the data represented in an existing `BitVector32` class.
public	`BitVector64(BitVector64 value)` Initializes a new instance of the `BitVector64` class containing the data represented in an existing `BitVector64` structure.

Method Summary

static long	`createMask()` Creates the first mask in a series of masks that can be used to retrieve individual bits in a `BitVector64` that is set up as bit flags.
static long	`createMask(long previous)` Creates an additional mask following the specified mask in a series of masks that can be used to retrieve individual bits in a `BitVector64` that is set up as bit flags.
static `BitVector64.Section`	`createSection(int maxValue)` Creates the first `BitVector64.Section` in a series of sections that contain small integers.
static `BitVector64.Section`	`createSection(int maxValue, BitVector64.Section previous)` Creates a new `BitVector64.Section` following the specified `BitVector64.Section` in a series of sections that contain integers.
boolean	`equals(Object obj)` Determines whether the specified object is equal to the `BitVector64`.

boolean	getBit(long bit) Gets the value of the specified bit flag or section.
long	getData() Gets the value of the BitVector64 as a long integer.
long	getValue(BitVector64.Section section) Gets the value stored in the specified BitVector64.Section.
int	hashCode() Serves as a hash function for the BitVector64.
void	setBit(long bit, boolean value) Sets the value of the specified bit flag or section.
void	setValue(BitVector64.Section section, long value) Sets the value stored in the specified BitVector64.Section.
String	toString() Returns a string that represents the current BitVector64.
static String	toString(BitVector64 value) Returns a string that represents the specified BitVector64.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

BitVector64

`public` **`BitVector64`**`(int data)`

Initializes a new instance of the BitVector64 class containing the data represented in an integer.

Parameters:
`data` - an integer representing the data of the new `BitVector64`.

BitVector64

`public` **`BitVector64`**`(long data)`

Initializes a new instance of the BitVector64 class containing the data represented in a long integer.

Parameters:
`data` - a long integer representing the data of the new `BitVector64`.

BitVector64

`public` **`BitVector64`**`(BitVector32 value)`

Initializes a new instance of the BitVector64 class containing the data represented in an existing BitVector32 class.

Parameters:

(continued from last page)

`value` - a `BitVector32` object that contains the data of the new `BitVector64`.

Throws:

`NullPointerException` - if `value` is `null` reference.

BitVector64

```
public BitVector64(BitVector64 value)
```

Initializes a new instance of the `BitVector64` class containing the data represented in an existing `BitVector64` structure.

Parameters:

`value` - a `BitVector64` object that contains the data of the new `BitVector64`.

Throws:

`NullPointerException` - if `value` is `null` reference.

Methods

getBit

```
public boolean getBit(long bit)
```

Gets the value of the specified bit flag or section.

Parameters:

`bit` - a mask that indicates the bit to get.

Returns:

`true` if the specified bit flag is on 1; otherwise, `false`.

setBit

```
public void setBit(long bit,
         boolean value)
```

Sets the value of the specified bit flag or section.

Parameters:

`bit` - bit a mask that indicates the bit to set.
`value` - `true` to set the specified bit flag to 1; otherwise, `false`.

getValue

```
public long getValue(BitVector64.Section section)
```

Gets the value stored in the specified `BitVector64.Section`.

Parameters:

`section` - a `BitVector64.Section` that contains the value to get or set.

Returns:

The value stored in the specified `BitVector64.Section`.

(continued from last page)

setValue

```
public void setValue(BitVector64.Section section,
         long value)
```

Sets the value stored in the specified `BitVector64.Section`.

Parameters:

`section` - a `BitVector64.Section` that contains the value to get or set.

`value` - the `long` value to be stored in the specified `BitVector64.Section`.

getData

```
public long getData()
```

Gets the value of the `BitVector64` as a `long` integer.

Returns:

The value of the `BitVector64` as an integer.

createMask

```
public static long createMask()
```

Creates the first mask in a series of masks that can be used to retrieve individual bits in a `BitVector64` that is set up as bit flags.

Returns:

A mask that isolates the first bit flag in the `BitVector64`.

createMask

```
public static long createMask(long previous)
  throws IllegalStateException
```

Creates an additional mask following the specified mask in a series of masks that can be used to retrieve individual bits in a `BitVector64` that is set up as bit flags.

Parameters:

`previous` - the mask that indicates the previous bit flag.

Returns:

A mask that isolates the bit flag following the one that `previous` points to in `BitVector64`.

Throws:

`IllegalStateException` - if `previous` indicates the last bit flag in the `BitVector64`.

createSection

```
public static BitVector64.Section createSection(int maxValue)
  throws IllegalArgumentException
```

Creates the first `BitVector64.Section` in a series of sections that contain small integers.

Parameters:

`maxValue` - a 32-bit signed integer that specifies the maximum value for the new `BitVector64.Section`.

Returns:

A `BitVector64.Section` that can hold a number from zero to `maxValue`.

(continued from last page)

Throws:
`IllegalArgumentException` - if `maxValue` is less than 1.

createSection

```
public static BitVector64.Section createSection(int maxValue,
          BitVector64.Section previous)
  throws IllegalArgumentException
```

Creates a new `BitVector64.Section` following the specified `BitVector64.Section` in a series of sections that contain integers.

Parameters:
`maxValue` - a 32-bit signed integer that specifies the maximum value for the new `BitVector64.Section`.
`previous` - the previous `BitVector64.Section` in the `BitVector64`.

Returns:
A `BitVector64.Section` that can hold a number from zero to `maxValue`.

Throws:
`IllegalArgumentException` - if `maxValue` is less than 1.
`IllegalStateException` - if

- `previous` includes the final bit in the `BitVector64`.
- `maxValue` is greater than the highest value that can be represented by the number of bits after previous.

equals

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

Determines whether the specified object is equal to the `BitVector64`.

Parameters:
`obj` - the object to compare with the current `BitVector64`.

Returns:
`true` if the specified object is equal to the `BitVector64`; otherwise, `false`.

hashCode

```
public int hashCode()
```

Serves as a hash function for the `BitVector64`.

The hash code of a `BitVector64` is based on the value of `getData()`. Two instances of `BitVector64` with the same value for `Data` will also generate the same hash code.

Returns:
A hash code for the `BitVector64`.

toString

```
public String toString()
```

Returns a string that represents the current `BitVector64`.

Returns:

(continued from last page)

A string that represents the current `BitVector64`.

toString

```
public static String toString(BitVector64 value)
```

Returns a string that represents the specified `BitVector64`.

Parameters:
`value` - the `BitVector64` to represent as string.

Returns:
A string that represents the specified `BitVector64`.

com.imis.util
Class BitVector64.Section

```
java.lang.Object
  |
  +-com.imis.util.BitVector64.Section
```

public static final class **BitVector64.Section**
extends Object

Represents a section of the vector that can contain a long integer.

Method Summary

`boolean`	`equals(BitVector64.Section obj)` Determines whether the specified `BitVector64.Section` object is the same as the current `BitVector64.Section` object.
`boolean`	`equals(Object obj)` Determines whether the specified object is the same as the current `BitVector64.Section` object.
`int`	`getMask()` Gets a mask that isolates this section within the `BitVector64`.
`int`	`getOffset()` Gets the offset of this section from the start of the `BitVector64`.
`int`	`hashCode()` Serves as a hash function for the current `BitVector64.Section`, suitable for hashing algorithms and data structures, such as a hash table.
`String`	`toString()` Returns a string that represents the current `BitVector64.Section`.
`static String`	`toString(BitVector64.Section value)` Returns a string that represents the specified `BitVector64.Section`.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

getMask

`public int getMask()`

Gets a mask that isolates this section within the `BitVector64`.

Returns:
A mask that isolates this section within the `BitVector64`.

getOffset

`public int getOffset()`

Gets the offset of this section from the start of the `BitVector64`.

Returns:
The offset of this section from the start of the `BitVector64`.

equals

`public boolean equals(BitVector64.Section obj)`

Determines whether the specified `BitVector64.Section` object is the same as the current `BitVector64.Section` object.

Parameters:
`obj` - the `BitVector64.Section` object to compare with the current `BitVector64.Section` object.

Returns:
`true` if the `obj` parameter is the same as the current `BitVector64.Section` object; otherwise `false`.

equals

`public boolean equals(Object obj)`

Determines whether the specified object is the same as the current `BitVector64.Section` object.

Returns:
`true` if the specified object is the same as the current `BitVector64.Section` object; otherwise, `false`.

hashCode

`public int hashCode()`

Serves as a hash function for the current `BitVector64.Section`, suitable for hashing algorithms and data structures, such as a hash table.

This method generates the same hash code for two objects that are equal according to the `equals(BitVector64.Section)` method.

Returns:
A hash code for the current `BitVector64.Section`.

toString

`public String toString()`

Returns a string that represents the current `BitVector64.Section`.

Returns:
A string that represents the current `BitVector64.Section`.

toString

`public static String toString(BitVector64.Section value)`

Returns a string that represents the specified `BitVector64.Section`.

(continued from last page)

Parameters:

`value` - the `BitVector64.Section` to represent as string.

Returns:

A string that represents the specified `BitVector64.Section`.

com.imis.util
Class Convert

```
java.lang.Object
  |
  +-com.imis.util.Convert
```

public final class **Convert**
extends Object

Provides methods for converting an array of bytes to and from a `String`.

Constructor Summary	
`public`	`Convert()`

Method Summary	
`static byte[]`	`base16ToBytes(String value)` Converts the specified string, which encodes binary data as base16 digits (hexadecimal), to an equivalent 8-bit unsigned integer array.
`static byte[]`	`base64ToBytes(String value)` Converts the specified string, which encodes binary data as base64 digits, to an equivalent 8 -bit unsigned integer array.
`static byte[]`	`base64UrlToBytes(String value)` Converts the specified string, which encodes binary data as base64url digits, to an equivalent array of bytes.
`static byte[]`	`base85ToBytes(String value)` Converts the specified string, which encodes binary data as base85 digits, to an equivalent 8 -bit unsigned integer array.
`static String`	`bytesToBase16(byte[] b)` Converts an array of bytes to its equivalent string representation encoded with base16 digits (hexadecimal).
`static String`	`bytesToBase16(byte[] b, int off, int len)` Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base16 digits (hexadecimal).
`static String`	`bytesToBase64(byte[] b)` Converts an array of bytes to its equivalent string representation encoded with base64 digits.
`static String`	`bytesToBase64(byte[] b, boolean padding)` Converts an array of bytes to its equivalent string representation encoded with base64 digits with optional padding.
`static String`	`bytesToBase64(byte[] b, int off, int len)` Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base64 digits.
`static String`	`bytesToBase64(byte[] b, int off, int len, boolean padding)` Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base64 digits with optional padding.

static String	`bytesToBase64Url(byte[] b)` Converts an array of bytes to its equivalent string representation encoded with base64url digits.
static String	`bytesToBase64Url(byte[] b, boolean padding)` Converts an array of bytes to its equivalent string representation encoded with base64url digits with optional padding.
static String	`bytesToBase64Url(byte[] b, int off, int len)` Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base64url digits.
static String	`bytesToBase64Url(byte[] b, int off, int len, boolean padding)` Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base64url digits with optional padding.
static String	`bytesToBase85(byte[] b)` Converts an array of bytes to its equivalent string representation encoded with base85 digits without padding.
static String	`bytesToBase85(byte[] b, boolean padding)` Converts an array of bytes to its equivalent string representation encoded with base85 digits with optional padding.
static String	`bytesToBase85(byte[] b, int off, int len)` Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base85 digits without padding.
static String	`bytesToBase85(byte[] b, int off, int len, boolean padding)` Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base85 digits with optional padding.
static String	`byteToHex(byte b)` Returns a string representation of the byte as an unsigned integer in base 16 with leading zeros.
static String	`intToHex(int i)` Returns a string representation of the integer as an unsigned integer in base 16 with leading zeros.
static String	`longToHex(long l)` Returns a string representation of the long integer as an unsigned long integer in base 16 with leading zeros.
static String	`shortToHex(short s)` Returns a string representation of the short as an unsigned integer in base 16 with leading zeros.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

Convert

```
public Convert()
```

(continued from last page)

Methods

byteToHex

`public static String **byteToHex**(byte b)`

Returns a string representation of the byte as an unsigned integer in base 16 with leading zeros.

Parameters:
`b` - a byte to be converted to a string.

Returns:
The string representation of the unsigned byte value in hexadecimal (base 16) with leading zeros.

---

shortToHex

`public static String **shortToHex**(short s)`

Returns a string representation of the short as an unsigned integer in base 16 with leading zeros.

Parameters:
`s` - a byte to be converted to a string.

Returns:
The string representation of the unsigned short value in hexadecimal (base 16) with leading zeros.

---

intToHex

`public static String **intToHex**(int i)`

Returns a string representation of the integer as an unsigned integer in base 16 with leading zeros.

Parameters:
`i` - an integer to be converted to a string.

Returns:
The string representation of the unsigned integer value in hexadecimal (base 16) with leading zeros.

---

longToHex

`public static String **longToHex**(long l)`

Returns a string representation of the long integer as an unsigned long integer in base 16 with leading zeros.

Parameters:
`l` - a long integer to be converted to a string.

Returns:
The string representation of the unsigned long integer value in hexadecimal (base 16) with leading zeros.

---

bytesToBase16

`public static String **bytesToBase16**(byte[] b)`

(continued from last page)

Converts an array of bytes to its equivalent string representation encoded with base16 digits (hexadecimal).

The returned string is twice the length of the input byte array, since every byte in the input array is converted to a two-digit hexadecimal. The output hex characters are upper case.

This method calls `bytesToBase64(byte[], int, int)`.

Parameters:

`b` - an array of bytes.

Returns:

The string representation, in base16, of the contents of `b`.

Throws:

`NullPointerException` - if `b` is a `null` reference.

bytesToBase16

```
public static String bytesToBase16(byte[] b,
         int off,
         int len)
```

Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base16 digits (hexadecimal).

The returned string is twice the length of the input byte array, since every byte in the input array is converted to a two-digit hexadecimal. The output hex characters are in lower case.

Parameters:

`b` - an array of bytes.
`off` - the zero-based byte offset in the array `b` at which to begin converting bytes.
`len` - the number of bytes to be converted.

Returns:

The string representation, in base16, of the contents of `b`.

Throws:

`NullPointerException` - if `b` is a `null` reference.
`IndexOutOfBoundsException` - if if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

base16ToBytes

```
public static byte[] base16ToBytes(String value)
  throws NumberFormatException
```

Converts the specified string, which encodes binary data as base16 digits (hexadecimal), to an equivalent 8-bit unsigned integer array.

This method is case insensitive and assumes zero padding when odd number of characters is supplied.

Parameters:

`value` - a string of hexadecimal characters.

Returns:

An array of bytes built from the bytes of the input string.

Throws:

`NumberFormatException` - if any character in the input string is not a valid hexadecimal digit.

(continued from last page)

bytesToBase64

```
public static String bytesToBase64(byte[] b)
```

Converts an array of bytes to its equivalent string representation encoded with base64 digits.

This method calls `bytesToBase64(byte[], int, int, boolean)` with `padding` parameter set to `false`.

Parameters:
`b` - an array of bytes.

Returns:
The string representation, in base64, of the contents of `b`.

Throws:
`NullPointerException` - if `b` is a `null` reference.

bytesToBase64

```
public static String bytesToBase64(byte[] b,
         boolean padding)
```

Converts an array of bytes to its equivalent string representation encoded with base64 digits with optional padding.

This method calls `bytesToBase64(byte[], int, int, boolean)`.

Padding character is equal to '='.

Parameters:
`b` - an array of bytes.
`padding` - `true` if padding is added; otherwise `false`.

Returns:
The string representation, in base64, of the contents of `b`.

Throws:
`NullPointerException` - if `b` is a `null` reference.

bytesToBase64

```
public static String bytesToBase64(byte[] b,
         int off,
         int len)
```

Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base64 digits.

This method calls `bytesToBase64(byte[], int, int, boolean)` with `padding` parameter set to `false`.

Parameters:
`b` - an array of bytes.
`off` - the zero-based byte offset in the array `b` at which to begin converting bytes.
`len` - the number of bytes to be converted.

Returns:
The string representation, in base64, of the contents of `b`.

Throws:
`NullPointerException` - if `b` is a `null` reference.
`IndexOutOfBoundsException` - if if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

bytesToBase64

```
public static String bytesToBase64(byte[] b,
                     int off,
                     int len,
                     boolean padding)
```

Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base64 digits with optional padding.

Padding character is equal to '='.

Parameters:

`b` - an array of bytes.

`off` - the zero-based byte offset in the array `b` at which to begin converting bytes.

`len` - the number of bytes to be converted.

`padding` - `true` if padding is added; otherwise `false`.

Returns:

The string representation, in base64, of the contents of `b`.

Throws:

`NullPointerException` - if `b` is a `null` reference.

`IndexOutOfBoundsException` - if if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

base64ToBytes

```
public static byte[] base64ToBytes(String value)
```

Converts the specified string, which encodes binary data as base64 digits, to an equivalent 8-bit unsigned integer array.

Parameters:

`value` - the string containing the base64 characters.

Returns:

An array of bytes equivalent to the specified string.

Throws:

`NullPointerException` - if `value` is a `null` reference.
`NumberFormatException` - if

- Invalid base64 character.
- Invalid base64 string.

bytesToBase64Url

```
public static String bytesToBase64Url(byte[] b)
```

Converts an array of bytes to its equivalent string representation encoded with base64url digits.

The base64url encoding is a base64 encoding in which the last two base64 digits are '-' and '_' instead of '+' and '/'. For more information see RFC 4648 `- The Base16, Base32, and Base64 Data Encodings`.

This method calls `bytesToBase64Url(byte[], int, int, boolean)` with `padding` parameter set to `false`.

Parameters:

`b` - an array of bytes.

(continued from last page)

Returns:

The string representation, in base64url, of the contents of `b`.

Throws:

`NullPointerException` - if `b` is a `null` reference.

bytesToBase64Url

```
public static String bytesToBase64Url(byte[] b,
          boolean padding)
```

Converts an array of bytes to its equivalent string representation encoded with base64url digits with optional padding.

This method calls `bytesToBase64Url(byte[], int, int, boolean)`.

Padding character is equal to '='.

Parameters:

`b` - an array of bytes.

`padding` - `true` if padding is added; otherwise `false`.

Returns:

The string representation, in base64url, of the contents of `b`.

Throws:

`NullPointerException` - if `b` is a `null` reference.

bytesToBase64Url

```
public static String bytesToBase64Url(byte[] b,
          int off,
          int len)
```

Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base64url digits.

The base64url encoding is a base64 encoding in which the last two base64 digits are '-' and '_' instead of '+' and '/'. For more information see RFC 4648 `- The Base16, Base32, and Base64 Data Encodings`.

This method calls `bytesToBase64Url(byte[], int, int, boolean)` with `padding` parameter set to `false`.

Parameters:

`b` - an array of bytes.

`off` - the zero-based byte offset in the array `b` at which to begin converting bytes.

`len` - the number of bytes to be converted.

Returns:

The string representation, in base64url, of the contents of `b`.

Throws:

`NullPointerException` - if `b` is a `null` reference.

`IndexOutOfBoundsException` - if if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

bytesToBase64Url

```
public static String bytesToBase64Url(byte[] b,
          int off,
          int len,
          boolean padding)
```

(continued from last page)

Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base64url digits with optional padding.

Padding character is equal to '='.

The base64url encoding is a base64 encoding in which the last two base64 digits are '-' and '_' instead of '+' and '/'. For more information see RFC 4648 `- The Base16, Base32, and Base64 Data Encodings`.

Parameters:

`b` - an array of bytes.
`off` - the zero-based byte offset in the array `b` at which to begin converting bytes.
`len` - the number of bytes to be converted.
`padding` - `true` if padding is added; otherwise `false`.

Returns:

The string representation, in base64url, of the contents of `b`.

Throws:

`NullPointerException` - if `b` is a `null` reference.
`IndexOutOfBoundsException` - if if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

base64UrlToBytes

`public static byte[] **base64UrlToBytes**(String value)`

Converts the specified string, which encodes binary data as base64url digits, to an equivalent array of bytes.

The base64url encoding is a base64 encoding in which the last two base64 digits are '-' and '_' instead of '+' and '/'. For more information see RFC 4648 `- The Base16, Base32, and Base64 Data Encodings`.

Parameters:

`value` - the string containing the modified base64url characters.

Returns:

An array of bytes equivalent to the specified string.

Throws:

`NullPointerException` - if `value` is a `null` reference.
`NumberFormatException` - if

- Invalid base64 character.
- Invalid base64 string.

bytesToBase85

`public static String **bytesToBase85**(byte[] b)`

Converts an array of bytes to its equivalent string representation encoded with base85 digits without padding.

This method calls `bytesToBase85(byte[], boolean)` with `padding` parameter set to `false`.

Parameters:

`b` - an array of bytes.

Returns:

The string representation, in base85, of the contents of `b`.

(continued from last page)

Throws:

`NullPointerException` - if `b` is a `null` reference.

bytesToBase85

```
public static String bytesToBase85(byte[] b,
         int off,
         int len)
```

Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base85 digits without padding.

This method calls `bytesToBase85(byte[], int, int, boolean)` with `padding` parameter set to `false`.

Parameters:

`b` - an array of bytes.
`off` - the zero-based byte offset in the array `b` at which to begin converting bytes.
`len` - the number of bytes to be converted.

Returns:

The string representation, in base85, of the contents of `b`.

Throws:

`NullPointerException` - if `b` is a `null` reference.
`IndexOutOfBoundsException` - if if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

bytesToBase85

```
public static String bytesToBase85(byte[] b,
         boolean padding)
```

Converts an array of bytes to its equivalent string representation encoded with base85 digits with optional padding.

This method calls `bytesToBase85(byte[], int, int, boolean)`.

Padding character is equal to '.'.

Parameters:

`b` - an array of bytes.
`padding` - `true` if padding is added; otherwise `false`.

Returns:

The string representation, in base85, of the contents of `b`.

Throws:

`NullPointerException` - if `b` is a `null` reference.

bytesToBase85

```
public static String bytesToBase85(byte[] b,
         int off,
         int len,
         boolean padding)
```

Converts the specified number of bytes at the specified offset in the array of bytes to its equivalent string representation encoded with base85 digits with optional padding.

Padding character is equal to '.'.

Parameters:

`b` - an array of bytes.

(continued from last page)

`padding` - `true` if padding is added; otherwise `false`.
`off` - the zero-based byte offset in the array `b` at which to begin converting bytes.
`len` - the number of bytes to be converted.

Returns:
The string representation, in base85, of the contents of `b`.

Throws:
`NullPointerException` - if `b` is a `null` reference.
`IndexOutOfBoundsException` - if if `off` or `len` is negative, or if `off+len` is is greater than the length of `b`.

base85ToBytes

`public static byte[] **base85ToBytes**(String value)`

Converts the specified string, which encodes binary data as base85 digits, to an equivalent 8-bit unsigned integer array.

Parameters:
`value` - the string containing the base85 characters.

Returns:
An array of bytes equivalent to the specified string.

Throws:
`NullPointerException` - if `value` is a `null` reference.
`NumberFormatException` - if

- Invalid base85 character.
- Invalid base85 string.

com.imis.util
Class Crypt

```
java.lang.Object
  |
  +-com.imis.util.Crypt
```

public final class **Crypt**
extends Object

Provides IMiS cryptography utility methods.

Method Summary

`static String`	`decrypt(String value, byte[] key, byte[] radix)` Decrypts the radix representation of the encrypted string value given the specified key and radix array.
`static String`	`encrypt(String value, byte[] key, byte[] radix)` Encrypts the string value given the specified key and radix array.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods

encrypt

```
public static String encrypt(String value,
         byte[] key,
         byte[] radix)
```

Encrypts the string value given the specified key and radix array.

Parameters:
`value` - the string value.
`key` - the key array.
`radix` - the radix array.

Returns:
The radix representation of the encrypted string value.

Throws:
`NullPointerException` - if `value`, `key` or `radix` is a `null` reference.

decrypt

```
public static String decrypt(String value,
         byte[] key,
         byte[] radix)
```

Decrypts the radix representation of the encrypted string value given the specified key and radix array.

(continued from last page)

Parameters:

`value` - the radix representation of the encrypted string value.
`key` - the key array.
`radix` - the radix array.

Returns:

The decrypted radix representation of the encrypted string value.

Throws:

`NullPointerException` - if `value`, `key` or `radix` is a `null` reference.

com.imis.util
Class Debugging

```
java.lang.Object
  |
  +-com.imis.util.Debugging
```

public class **Debugging**
extends Object

Provides methods for sending strings to the debugger for display.

Constructor Summary	
public	`Debugging()`

Method Summary	
static int	`getPrintLevel()` Gets the print level of object fields.
static void	`printError(Throwable e)` Prints a stack trace of the specified error.
static void	`printLine()` Prints a new line to the standard output stream.
static void	`printLine(String output)` Prints a string to the standard output stream.
static void	`printLine(String output, Object arg0)` Prints a formatted string to the standard output stream.
static void	`printLine(String output, Object[] args)` Prints a formatted string to the standard output stream.
static void	`printObject(Object obj)` Prints the object public property values up to default level to the standard output stream.
static void	`printObject(Object obj, int level)` Prints the object public property values up to the specified level to the standard output stream.
static void	`printObject(Object obj, String name)` Prints the object public property values up to default level using a custom object name to the standard output stream.
static void	`printObject(Object obj, String name, int level)` Prints the object public property values up to the specified level using a custom object name to the standard output stream.
static void	`setPrintLevel(int value)` Sets the print level of object fields.

Methods inherited from class `java.lang.Object`

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
wait
```

Constructors

Debugging

```
public Debugging()
```

Methods

getPrintLevel

```
public static int getPrintLevel()
```

Gets the print level of object fields.

Returns:
The print level of object fields.

setPrintLevel

```
public static void setPrintLevel(int value)
```

Sets the print level of object fields.

Parameters:
`value` - the print level of object fields.

printLine

```
public static void printLine()
```

Prints a new line to the standard output stream.

printLine

```
public static void printLine(String output)
```

Prints a string to the standard output stream.

Parameters:
`output` - a string to output.

printLine

```
public static void printLine(String output,
         Object arg0)
```

Prints a formatted string to the standard output stream.

Parameters:
`output` - a string to output containing one format item.

(continued from last page)

`arg0` - an object to format.

printLine

```
public static void printLine(String output,
         Object[] args)
```

Prints a formatted string to the standard output stream.

Parameters:

`output` - a string to output containing zero or more format items.
`args` - an object array containing zero or more objects to format.

printObject

```
public static void printObject(Object obj)
```

Prints the object public property values up to default level to the standard output stream.

Parameters:

`obj` - an object.

printObject

```
public static void printObject(Object obj,
         int level)
```

Prints the object public property values up to the specified level to the standard output stream.

Parameters:

`obj` - an object.
`level` - a print level.

printObject

```
public static void printObject(Object obj,
         String name)
```

Prints the object public property values up to default level using a custom object name to the standard output stream.

Parameters:

`obj` - an object.
`name` - an object name.

printObject

```
public static void printObject(Object obj,
         String name,
         int level)
```

Prints the object public property values up to the specified level using a custom object name to the standard output stream.

Parameters:

`obj` - an object.
`name` - an object name.
`level` - a print level.

(continued from last page)

printError

```
public static void printError(Throwable e)
```

Prints a stack trace of the specified error.

Parameters:
`e` - an error.

com.imis.util
Class HashBiMap

```
java.lang.Object
  |
  +-com.imis.util.HashBiMap
```

All Implemented Interfaces:
IReadOnlyMap, Map

public class **HashBiMap**
extends Object
implements Map, IReadOnlyMap

Defines a map that allows bidirectional lookup between key and values.
Parameters:
`K` - the type of the keys in the map., `V` - the type of the values in the map.

Constructor Summary

`public`	`HashBiMap()` Initializes a new instance of the `HashBiMap` class.
`public`	`HashBiMap(Map m)` Initializes a new instance of the `HashBiMap` class from the specified map.

Method Summary

`void`	`clear()` Removes all mappings from this bidirectional map.
`boolean`	`containsKey(Object key)` Returns `true` if this bidirectional map contains a mapping for the specified key.
`boolean`	`containsValue(Object value)` Returns `true` if this bidirectional map maps one or more keys to this value.
`Set`	`entrySet()` Returns a set view of the mappings contained in this bidirectional map.
`boolean`	`equals(Object o)` Compares the specified object with this bidirectional map for equality.
`Object`	`get(Object key)` Returns the value to which this map maps the specified key or a `null` reference, if the map contains no mapping for this key.
`int`	`hashCode()` Returns the hash code value for this bidirectional map.
`Map`	`inverseBiMap()` Returns the inverse view of this bidirectional map where the keys and values are reversed.
`boolean`	`isEmpty()` Returns `true` if this bidirectional map contains no key-value mappings.

`Set`	`keySet()` Returns a set view of the keys contained in this bidirectional map.
`Object`	`put(Object key, Object value)` Associates the specified value with the specified key in this map.
`void`	`putAll(Map m)` Copies all of the mappings from the specified map to this bidirectional map.
`Object`	`remove(Object key)` Removes the mapping for this key from this bidirectional map if present.
`int`	`size()` Returns the number of key-value mappings in this bidirectional map.
`String`	`toString()` Returns a string representation of this bidirectional map.
`Collection`	`values()` Returns a collection view of the values contained in this bidirectional map.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.util.Map`

`clear, containsKey, containsValue, entrySet, equals, get, hashCode, isEmpty, keySet, put, putAll, remove, size, values`

Methods inherited from interface `com.imis.util.IReadOnlyMap`

`containsKey, containsValue, entrySet, equals, get, hashCode, isEmpty, keySet, size, values`

Constructors

HashBiMap

`public HashBiMap()`

Initializes a new instance of the `HashBiMap` class.

HashBiMap

`public HashBiMap(Map m)`

Initializes a new instance of the `HashBiMap` class from the specified map.

Parameters:
`m` - the map whose mappings are to be placed in this bidirectional map.

Methods

(continued from last page)

inverseBiMap

`public Map` **`inverseBiMap`**`()`

Returns the inverse view of this bidirectional map where the keys and values are reversed.

Returns:
the inverse view of this bidirectional map where the keys and values are reversed.

size

`public int` **`size`**`()`

Returns the number of key-value mappings in this bidirectional map.

Returns:
the number of key-value mappings in this bidirectional map.

isEmpty

`public boolean` **`isEmpty`**`()`

Returns `true` if this bidirectional map contains no key-value mappings.

Returns:
`true` if this bidirectional map contains no key-value mappings; otherwise `false`.

containsKey

`public boolean` **`containsKey`**`(Object key)`

Returns `true` if this bidirectional map contains a mapping for the specified key.

Parameters:
`key` - the key whose presence in this map is to be tested.

Returns:
`true` if this bidirectional map contains a mapping for the specified key; otherwise `false`.

containsValue

`public boolean` **`containsValue`**`(Object value)`

Returns `true` if this bidirectional map maps one or more keys to this value.

Parameters:
`value` - the value whose presence in this map is to be tested.

Returns:
`true` if this bidirectional map maps one or more keys to this value; otherwise `false`.

get

`public Object` **`get`**`(Object key)`

(continued from last page)

Returns the value to which this map maps the specified key or a `null` reference, if the map contains no mapping for this key.

A return value of `null` does not *necessarily* indicate that the map contains no mapping for the key; it's also possible that the map explicitly maps the key to `null`. The `containsKey()` operation may be used to distinguish these two cases.

Parameters:
`key` - the key whose associated value is to be returned.

Returns:
The value to which this bidirectional map maps the specified key.

See Also:
`containsKey(Object)`

put

```
public Object put(Object key,
                  Object value)
```

Associates the specified value with the specified key in this map. If the map previously contained a mapping for this key, the old value is replaced.

Parameters:
`key` - the key with which the specified value is to be associated.
`value` - the value to be associated with the specified key.

Returns:
Previous value associated with specified key, or `null` if there was no mapping for key. A `null` return can also indicate that the bidirectional map previously associated `null` with the specified key.

remove

```
public Object remove(Object key)
```

Removes the mapping for this key from this bidirectional map if present.

Parameters:
`key` - the key whose mapping is to be removed from the map.

Returns:
Previous value associated with specified key or `null` if there was no mapping for key. A `null` return can also indicate that the map previously associated `null` with the specified key.

putAll

```
public void putAll(Map m)
```

Copies all of the mappings from the specified map to this bidirectional map. These mappings will replace any mappings that this map had for any of the keys currently in the specified map.

Parameters:
`m` - mappings to be stored in this bidirectional map.

Throws:
`NullPointerException` - if the specified map is `null`.

clear

```
public void clear()
```

(continued from last page)

Removes all mappings from this bidirectional map.

keySet

`public Set **keySet**()`

Returns a set view of the keys contained in this bidirectional map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own remove operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll` `retainAll` and `clear` operations. It does not support the `add` or `addAll` operations.

Returns:
a set view of the keys contained in this bidirectional map.

values

`public Collection **values**()`

Returns a collection view of the values contained in this bidirectional map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress (except through the iterator's own `remove` operation), the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Collection.remove`, `removeAll`, `retainAll` and `clear` operations. It does not support the `add` or `addAll` operations.

Returns:
a collection view of the values contained in this map.

entrySet

`public Set **entrySet**()`

Returns a set view of the mappings contained in this bidirectional map. Each element in the returned set is a `Map.Entry`. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own `remove` operation, or through the `setValue` operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll`, `retainAll` and `clear` operations. It does not support the `add` or `addAll` operations.

Returns:
a set view of the mappings contained in this bidirectional map.

equals

`public boolean **equals**(Object o)`

Compares the specified object with this bidirectional map for equality. Returns `true` if the given object is also a map and the two maps represent the same mappings. More formally, two maps `t1` and `t2` represent the same mappings if `t1.entrySet().equals(t2.entrySet())`. This ensures that the `equals` method works properly across different implementations of the `Map` interface.

Parameters:
`o` - the object to be compared for equality with this map.

Returns:
`true` if the specified object is equal to this map; otherwise `false`.

hashCode

`public int **hashCode**()`

(continued from last page)

Returns the hash code value for this bidirectional map. The hash code of a map is defined to be the sum of the hash codes of each entry in the map's `entrySet` view. This ensures that `t1.equals(t2)` implies that `t1.hashCode()==t2.hashCode()` for any two maps `t1` and `t2`, as required by the general contract of `Object.hashCode`.

Returns:
the hash code value for this bidirectional map.

toString

`public String **toString**()`

Returns a string representation of this bidirectional map. The string representation consists of a list of key-value mappings in the order returned by the map's `entrySet` view's iterator, enclosed in braces (`"{}"`). Adjacent mappings are separated by the characters `", "` (comma and space). Each key-value mapping is rendered as the key followed by an equals sign (`"="`) followed by the associated value. Keys and values are converted to strings as by `String.valueOf(Object)`.

This implementation creates an empty string buffer, appends a left brace, and iterates over the map's `entrySet` view, appending the string representation of each `map.entry` in turn. After appending each entry except the last, the string `", "` is appended. Finally a right brace is appended. A string is obtained from the string buffer, and returned.

Returns:
a String representation of this bidirectional map.

com.imis.util
Interface ILargeReadOnlyCollection

All Subinterfaces:
ILargeReadOnlyList, ILargeReadOnlySet

public interface **ILargeReadOnlyCollection**
extends Iterable

Provides a read-only version of the `Collection` interface with the number of elements limited with `long` type.

A `ILargeReadOnlyCollection` is simply a `Collection` without methods that allow changes in the collection.
See Also:
`java.util.Collection`

Method Summary

`boolean`	`contains(Object o)` Determines if this read-only collection contains the specified element.
`boolean`	`containsAll(Collection c)` Determines if this read-only collection contains all of the elements in the specified collection.
`boolean`	`equals(Object o)` Compares the specified object with this read-only collection for equality.
`int`	`hashCode()` Returns the hash code value for this read-only collection.
`boolean`	`isEmpty()` Determines if this read-only collection contains no elements.
`Iterator`	`iterator()` Returns an iterator over the elements in this read-only collection.
`long`	`size()` Returns the number of elements in this read-only collection.
`Object[]`	`toArray()` Returns an array containing all of the elements in this read-only collection.
`Object[]`	`toArray(Object[] a)` Returns an array containing all of the elements in this read-only collection.

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods

(continued from last page)

size

`public long size()`

Returns the number of elements in this read-only collection.

If this collection contains more than `Long.MAX_VALUE` elements, returns `Long.MAX_VALUE`.

Returns:
The number of elements in this read-only collection.

---

isEmpty

`public boolean isEmpty()`

Determines if this read-only collection contains no elements.

Returns:
`true` if this read-only collection contains no elements; otherwise `false`.

---

contains

`public boolean contains(Object o)`

Determines if this read-only collection contains the specified element.

More formally, returns `true` if and only if this collection contains at least one element `e` such that `(null == o) ? (null == e) : o.equals(e)`.

Parameters:
`o` - the element whose presence in this collection is to be tested.

Returns:
`true` if this read-only collection contains the specified element; otherwise `false`.

Throws:
`ClassCastException` - if the type of the specified element is incompatible with this collection (optional).
`NullPointerException` - if the specified element is a `null` reference and this collection does not support `null` elements (optional).

---

iterator

`public Iterator iterator()`

Returns an iterator over the elements in this read-only collection.

There are no guarantees concerning the order in which the elements are returned (unless this collection is an instance of some class that provides a guarantee).

Note that `remove` operation is not supported by this iterator.

Returns:
An `Iterator` over the elements in this read-only collection.

---

toArray

`public Object[] toArray()`

(continued from last page)

Returns an array containing all of the elements in this read-only collection.

If the collection makes any guarantees as to what order its elements are returned by its iterator, this method must return the elements in the same order.

The returned array will be "safe" in that no references to it are maintained by this collection. (In other words, this method must allocate a new array even if this collection is backed by an array). The caller is thus free to modify the returned array.

This method acts as bridge between array-based and collection-based APIs.

Returns:
An array containing all of the elements in this read-only collection.

Throws:
`UnsupportedOperationException` - if collection to big for an array.

toArray

`public Object[] **toArray**(Object[] a)`

Returns an array containing all of the elements in this read-only collection.

The runtime type of the returned array is that of the specified array. If the collection fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this collection.

If this collection fits in the specified array with room to spare (i.e., the array has more elements than this collection), the element in the array immediately following the end of the collection is set to `null`. This is useful in determining the length of this collection *only* if the caller knows that this collection does not contain any `null` elements.)

If this collection makes any guarantees as to what order its elements are returned by its iterator, this method must return the elements in the same order.

Like the `toArray` method, this method acts as bridge between array-based and collection-based APIs.

Further, this method allows precise control over the runtime type of the output array, and may, under certain circumstances, be used to save allocation costs

Suppose `l` is a `List` known to contain only strings. The following code can be used to dump the list into a newly allocated array of `String`:

```
String[] x = (String[])l.toArray(new String[0]);
```

Note that `toArray(new Object[0])` is identical in function to `toArray()`.

Parameters:
`a` - the array into which the elements of this collection are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:
An array containing the elements of this read-only collection.

Throws:
`ArrayStoreException` - the runtime type of the specified array is not a supertype of the runtime type of every element in this collection.
`NullPointerException` - if the specified array is a `null` reference.
`UnsupportedOperationException` - if collection to big for an array.

(continued from last page)

containsAll

`public boolean **containsAll**(Collection c)`

Determines if this read-only collection contains all of the elements in the specified collection.

Parameters:
`c` - the collection to be checked for containment in this collection.

Returns:
`true` if this read-only collection contains all of the elements in the specified collection; otherwise `false`.

Throws:
`ClassCastException` - if the types of one or more elements in the specified collection are incompatible with this collection (optional).
`NullPointerException` - if the specified collection contains one or more `null` elements and this collection does not support `null` elements (optional).
`NullPointerException` - if the specified collection is a `null` reference.

See Also:
`contains(Object)`

equals

`public boolean **equals**(Object o)`

Compares the specified object with this read-only collection for equality.

While the `ILargeReadOnlyCollection` interface adds no stipulations to the general contract for the `Object.equals`, programmers who implement the `ILargeReadOnlyCollection` interface "directly" (in other words, create a class that is a `ILargeReadOnlyCollection` but is not `ILargeReadOnlySet` or `ILargeReadOnlyList`) must exercise care if they choose to override the `Object.equals`. It is not necessary to do so, and the simplest course of action is to rely on `Object`'s implementation, but the implementer may wish to implement a "value comparison" in place of the default "reference comparison." (The `ILargeReadOnlySet` and `ILargeReadOnlyList` interface mandate such value comparisons.)

The general contract for the `Object.equals` method states that equals must be symmetric (in other words, `a.equals(b)` if and only if `b.equals(a)`). The contracts for `ILargeReadOnlySet.equals` and `ILargeReadOnlyList.equals` state that lists are only equal to other lists, and sets to other sets. Thus, a custom `equals` method for a collection class that implements neither the `ILargeReadOnlySet` nor `ILargeReadOnlyList` interface must return `false` when this collection is compared to any list or set. (By the same logic, it is not possible to write a class that correctly implements both the `Set` and `List` interfaces.)

Parameters:
`o` - the `Object` to be compared for equality with this collection.

Returns:
`true` if the specified object is equal to this read-only collection; otherwise `false`.

See Also:
`Object.equals(java.lang.Object)`
`ILargeReadOnlySet.equals(Object)`
`ILargeReadOnlyList.equals(Object)`

hashCode

`public int **hashCode**()`

(continued from last page)

Returns the hash code value for this read-only collection.

While the `ILargeReadOnlyCollection` interface adds no stipulations to the general contract for the `Object.hashCode` method, programmers should take note that any class that overrides the `Object.equals` method must also override the `Object.hashCode` method in order to satisfy the general contract for the `Object.hashCode`method. In particular, `c1.equals(c2)` implies that `c1.hashCode() == c2.hashCode()`.

Returns:

The hash code value for this read-only collection.

See Also:

`Object.hashCode()`

`Object.equals(java.lang.Object)`

com.imis.util
Interface ILargeReadOnlyList

All Superinterfaces:

ILargeReadOnlyCollection

public interface **ILargeReadOnlyList**
extends ILargeReadOnlyCollection

Provides a read-only version of the `List` interface with the number of elements limited with `long` type.

A `ILargeReadOnlyList` is simply a `List` without methods that allow changes in the list.
See Also:
`java.util.List`

Method Summary

`boolean`	`contains(Object o)` Determines if this read-only list contains the specified element.
`boolean`	`containsAll(Collection c)` Determines if this read-only list contains all of the elements in the specified collection.
`boolean`	`equals(Object o)` Compares the specified object with this read-only list for equality.
`Object`	`get(long index)` Returns the element at the specified position in this read-only list.
`int`	`hashCode()` Returns the hash code value for this read-only list.
`long`	`indexOf(Object o)` Returns the index in this read-only list of the first occurrence of the specified element, or `-1` if this read-only list does not contain this element.
`boolean`	`isEmpty()` Determines if this read-only list contains no elements.
`Iterator`	`iterator()` Returns an iterator over the elements in this read-only list in proper sequence.
`long`	`lastIndexOf(Object o)` Returns the index in this read-only list of the last occurrence of the specified element, or `-1` if this read-only list does not contain this element.
`ListIterator`	`listIterator()` Returns a list iterator of the elements in this read-only list (in proper sequence).
`ListIterator`	`listIterator(long index)` Returns a list iterator of the elements in this read-only list (in proper sequence), starting at the specified position in this read-only list.
`long`	`size()` Returns the number of elements in this read-only list.

`ILargeReadOnlyList`	`subList(long fromIndex, long toIndex)` Returns a view of the portion of this read-only list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.
`Object[]`	`toArray()` Returns an array containing all of the elements in this read-only list in proper sequence.
`Object[]`	`toArray(Object[] a)` Returns an array containing all of the elements in this read-only list in proper sequence.

Methods inherited from interface `com.imis.util.ILargeReadOnlyCollection`

`contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `size`, `toArray`, `toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods

size

`public long size()`

Returns the number of elements in this read-only list.

Returns:
The number of elements in this read-only list.

isEmpty

`public boolean isEmpty()`

Determines if this read-only list contains no elements.

Returns:
`true` if this read-only list contains no elements; otherwise `false`.

contains

`public boolean contains(Object o)`

Determines if this read-only list contains the specified element.

More formally, returns `true` if and only if this list contains at least one element `e` such that `(null == o) ? (null == e) : o.equals(e)`.

Parameters:
`o` - the element whose presence in this list is to be tested.

Returns:
`true` if this read-only list contains the specified element; otherwise `false`.

Throws:
`ClassCastException` - if the type of the specified element is incompatible with this list (optional).
`NullPointerException` - if the specified element is a `null` reference and this list does not support `null` elements (optional).

iterator

`public Iterator` **`iterator`**`()`

Returns an iterator over the elements in this read-only list in proper sequence.

Note that `remove` operation is not supported by this iterator.

Returns:
An `Iterator` over the elements in this read-only list in proper sequence.

toArray

`public Object[]` **`toArray`**`()`

Returns an array containing all of the elements in this read-only list in proper sequence.

This method obeys the general contract of the `Collection.toArray` method.

Returns:
An array containing all of the elements in this read-only list in proper sequence.

Throws:
`UnsupportedOperationException` - if collection to big for an array.

See Also:
`Arrays#asList(Object[])`

toArray

`public Object[]` **`toArray`**`(Object[] a)`

Returns an array containing all of the elements in this read-only list in proper sequence.

The runtime type of the returned array is that of the specified array.

This method obeys the general contract of the `Collection.toArray(Object[])` method.

Parameters:
`a` - the array into which the elements of this list are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:
An array containing the elements of this read-only list.

Throws:
`ArrayStoreException` - if the runtime type of the specified array is not a supertype of the runtime type of every element in this list.
`NullPointerException` - if the specified array is a `null` reference.
`UnsupportedOperationException` - if collection to big for an array.

containsAll

`public boolean` **`containsAll`**`(Collection c)`

Determines if this read-only list contains all of the elements in the specified collection.

Parameters:
`c` - the collection to be checked for containment in this list.

(continued from last page)

Returns:

`true` if this read-only list contains all of the elements in the specified collection; otherwise `false`.

Throws:

`ClassCastException` - if the types of one or more elements in the specified collection are incompatible with this list (optional).
`NullPointerException` - if the specified collection contains one or more `null` elements and this list does not support `null` elements (optional).
`NullPointerException` - if the specified collection is a `null` reference.

See Also:

`contains(Object)`

equals

`public boolean **equals**(Object o)`

Compares the specified object with this read-only list for equality.

Returns `true` if and only if the specified object is also a list, both lists have the same size, and all corresponding pairs of elements in the two lists are equal. (Two elements `e1` and `e2` are equal if `(null == e1) ? (null == e2) : e1.equals(e2)`.) In other words, two lists are defined to be equal if they contain the same elements in the same order. This definition ensures that the equals method works properly across different implementations of the `List` interface.

Parameters:

`o` - the object to be compared for equality with this list.

Returns:

`true` if the specified object is equal to this read-only list; otherwise `false`.

hashCode

`public int **hashCode**()`

Returns the hash code value for this read-only list.

The hash code of a list is defined to be the result of the following calculation:

```
hashCode = 1;
 Iterator i = list.iterator();
 while (i.hasNext()) {
   Object obj = i.next();
   hashCode = 31 * hashCode + ((null == obj) ? 0 : obj.hashCode());
 }
```

This ensures that `list1.equals(list2)` implies that `list1.hashCode() == list2.hashCode()` for any two lists, `list1` and `list2`, as required by the general contract of `Object.hashCode`.

Returns:

The hash code value for this read-only list.

See Also:

`Object.hashCode()`
`Object.equals(java.lang.Object)`
`equals(Object)`

get

`public Object **get**(long index)`

Returns the element at the specified position in this read-only list.

Parameters:
`index` - the index of element to return.

Returns:
The element at the specified position in this read-only list.

Throws:
`IndexOutOfBoundsException` - if the index is out of range `(0 > index) || (index >= size())`.

indexOf

`public long **indexOf**(Object o)`

Returns the index in this read-only list of the first occurrence of the specified element, or `-1` if this read-only list does not contain this element.

More formally, returns the lowest index `i` such that `(null == o) ? (null == get(i)) : o.equals(get(i)))`, or `-1` if there is no such index.

Parameters:
`o` - the element to search for.

Returns:
The index in this read-only list of the first occurrence of the specified element, or `-1` if this read-only list does not contain this element.

Throws:
`ClassCastException` - if the type of the specified element is incompatible with this list (optional).
`NullPointerException` - if the specified element is a `null` reference and this list does not support `null` elements (optional).

lastIndexOf

`public long **lastIndexOf**(Object o)`

Returns the index in this read-only list of the last occurrence of the specified element, or `-1` if this read-only list does not contain this element.

More formally, returns the highest index `i` such that `(null == o) ? (null == get(i)) : o.equals(get(i)))`, or `-1` if there is no such index.

Parameters:
`o` - the element to search for.

Returns:
The index in this read-only list of the last occurrence of the specified element, or `-1` if this read-only list does not contain this element.

Throws:
`ClassCastException` - if the type of the specified element is incompatible with this list (optional).
`NullPointerException` - if the specified element is a `null` reference and this list does not support `null` elements (optional).

(continued from last page)

listIterator

`public ListIterator **listIterator**()`

Returns a list iterator of the elements in this read-only list (in proper sequence).

Note that modification operations are not supported by this list iterator.

Returns:
A list iterator of the elements in this read-only list (in proper sequence).

---

listIterator

`public ListIterator **listIterator**(long index)`

Returns a list iterator of the elements in this read-only list (in proper sequence), starting at the specified position in this read-only list.

The specified index indicates the first element that would be returned by an initial call to the `next` method. An initial call to the `previous` method would return the element with the specified index minus one.

Note that modification operations are not supported by this list iterator.

Parameters:
`index` - index of first element to be returned from the list iterator (by a call to the `next` method).

Returns:
A list iterator of the elements in this read-only list (in proper sequence), starting at the specified position in this read-only list.

Throws:
`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index > size()`).

---

subList

```
public ILargeReadOnlyList subList(long fromIndex,
    long toIndex)
```

Returns a view of the portion of this read-only list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive. If `fromIndex` and `toIndex` are equal, the returned list is empty.

The returned list is backed by this read-only list and supports only operations that do not change the list.

Parameters:
`fromIndex` - a low endpoint (inclusive) of the subList.
`toIndex` - a high endpoint (exclusive) of the subList.

Returns:
A view of the specified range within this read-only list.

Throws:
`IndexOutOfBoundsException` - for an illegal endpoint index value (`fromIndex < 0 || toIndex > size || fromIndex > toIndex`).

com.imis.util
Interface ILargeReadOnlySet

All Superinterfaces:
ILargeReadOnlyCollection

public interface **ILargeReadOnlySet**
extends ILargeReadOnlyCollection

Provides a read-only version of the `Set` interface.

A `ILargeReadOnlySet` is simply a `Set` without methods that allow changes in the set.
See Also:
`java.util.Set`

Method Summary

`boolean`	`contains(Object o)` Determines if this read-only set contains the specified element.
`boolean`	`containsAll(Collection c)` Determines if this read-only set contains all of the elements in the specified collection.
`boolean`	`equals(Object o)` Compares the specified object with this read-only set for equality.
`int`	`hashCode()` Returns the hash code value for this read-only set.
`boolean`	`isEmpty()` Determines if this read-only set contains no elements.
`Iterator`	`iterator()` Returns an iterator over the elements in this read-only set.
`long`	`size()` Returns the number of elements in this read-only set.
`Object[]`	`toArray()` Returns an array containing all of the elements in this read-only set.
`Object[]`	`toArray(Object[] a)` Returns an array containing all of the elements in this read-only set.

Methods inherited from interface `com.imis.util.ILargeReadOnlyCollection`

`contains, containsAll, equals, hashCode, isEmpty, iterator, size, toArray, toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods

(continued from last page)

size

`public long size()`

Returns the number of elements in this read-only set.

If this set contains more than `Long.MAX_VALUE` elements, returns `Long.MAX_VALUE`.

Returns:
The number of elements in this read-only set.

---

isEmpty

`public boolean isEmpty()`

Determines if this read-only set contains no elements.

Returns:
`true` if this read-only set contains no elements; otherwise `false`.

---

contains

`public boolean contains(Object o)`

Determines if this read-only set contains the specified element.

More formally, returns `true` if and only if this set contains an element `e` such that `(null == o) ? (null == e) : o.equals(e)`.

Parameters:
`o` - the element whose presence in this set is to be tested.

Returns:
`true` if this read-only set contains the specified element; otherwise `false`.

Throws:
`ClassCastException` - if the type of the specified element is incompatible with this set (optional).
`NullPointerException` - if the specified element is a `null` reference and this set does not support `null` elements (optional).

---

iterator

`public Iterator iterator()`

Returns an iterator over the elements in this read-only set.

The elements are returned in no particular order (unless this set is an instance of some class that provides a guarantee).

Note that `remove` operation is not supported by this iterator.

Returns:
An `Iterator` over the elements in this read-only set.

---

toArray

`public Object[] toArray()`

Returns an array containing all of the elements in this read-only set.

Obeys the general contract of the `Collection.toArray` method.

(continued from last page)

Returns:
An array containing all of the elements in this read-only set.

toArray

`public Object[] toArray(Object[] a)`

Returns an array containing all of the elements in this read-only set.

The runtime type of the returned array is that of the specified array.

Obeys the general contract of the `Collection.toArray(Object[])` method.

Parameters:
`a` - the array into which the elements of this set are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:
An array containing the elements of this read-only set.

Throws:
`ArrayStoreException` - the runtime type of the specified array is not a super type of the runtime type of every element in this set.
`NullPointerException` - if the specified array is a `null` reference.

containsAll

`public boolean containsAll(Collection c)`

Determines if this read-only set contains all of the elements in the specified collection.

If the specified collection is also a set, this method returns `true` if it is a subset of this set.

Parameters:
`c` - the collection to be checked for containment in this set.

Returns:
`true` if this read-only set contains all of the elements in the specified collection; otherwise `false`.

Throws:
`ClassCastException` - if the types of one or more elements in the specified collection are incompatible with this set (optional).
`NullPointerException` - if the specified collection contains one or more `null` elements and this collection does not support `null` elements (optional).
`NullPointerException` - if the specified collection is a `null` reference.

See Also:
`contains(Object)`

equals

`public boolean equals(Object o)`

Compares the specified object with this read-only set for equality.

Returns `true` if the specified object is also a set, the two sets have the same size, and every member of the specified set is contained in this set (or equivalently, every member of this set is contained in the specified set). This definition ensures that the equals method works properly across different implementations of the set interface.

Parameters:
`o` - the `Object` to be compared for equality with this set.

(continued from last page)

Returns:
`true` if the specified object is equal to this read-only set; otherwise `false`.

---

hashCode

`public int **hashCode**()`

Returns the hash code value for this read-only set.

The hash code of a set is defined to be the sum of the hash codes of the elements in the set, where the hashcode of a `null` element is defined to be zero. This ensures that `s1.equals(s2)` implies that `s1.hashCode()==s2.hashCode()` for any two sets `s1` and `s2`, as required by the general contract of the `Object.hashCode` method.

Returns:
The hash code value for this read-only set.

See Also:
`Object.hashCode()`
`Object.equals(java.lang.Object)`
`equals(Object)`

com.imis.util
Interface IReadOnlyCollection

All Subinterfaces:
IReadOnlyList, IReadOnlySet

public interface **IReadOnlyCollection**
extends Iterable

Provides a read-only version of the `Collection` interface.

A `IReadOnlyCollection` is simply a `Collection` without methods that allow changes in the collection.
See Also:
`java.util.Collection`

Method Summary

`boolean`	`contains(Object o)` Determines if this read-only collection contains the specified element.
`boolean`	`containsAll(Collection c)` Determines if this read-only collection contains all of the elements in the specified collection.
`boolean`	`equals(Object o)` Compares the specified object with this read-only collection for equality.
`int`	`hashCode()` Returns the hash code value for this read-only collection.
`boolean`	`isEmpty()` Determines if this read-only collection contains no elements.
`Iterator`	`iterator()` Returns an iterator over the elements in this read-only collection.
`int`	`size()` Returns the number of elements in this read-only collection.
`Object[]`	`toArray()` Returns an array containing all of the elements in this read-only collection.
`Object[]`	`toArray(Object[] a)` Returns an array containing all of the elements in this read-only collection.

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods

(continued from last page)

size

`public int size()`

Returns the number of elements in this read-only collection.

If this collection contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.

Returns:
The number of elements in this read-only collection.

isEmpty

`public boolean isEmpty()`

Determines if this read-only collection contains no elements.

Returns:
`true` if this read-only collection contains no elements; otherwise `false`.

contains

`public boolean contains(Object o)`

Determines if this read-only collection contains the specified element.

More formally, returns `true` if and only if this collection contains at least one element `e` such that `(null == o) ? (null == e) : o.equals(e)`.

Parameters:
`o` - the element whose presence in this collection is to be tested.

Returns:
`true` if this read-only collection contains the specified element; otherwise `false`.

Throws:
`ClassCastException` - if the type of the specified element is incompatible with this collection (optional).
`NullPointerException` - if the specified element is a `null` reference and this collection does not support `null` elements (optional).

iterator

`public Iterator iterator()`

Returns an iterator over the elements in this read-only collection.

There are no guarantees concerning the order in which the elements are returned (unless this collection is an instance of some class that provides a guarantee).

Note that `remove` operation is not supported by this iterator.

Returns:
An `Iterator` over the elements in this read-only collection.

toArray

`public Object[] toArray()`

(continued from last page)

Returns an array containing all of the elements in this read-only collection.

If the collection makes any guarantees as to what order its elements are returned by its iterator, this method must return the elements in the same order.

The returned array will be "safe" in that no references to it are maintained by this collection. (In other words, this method must allocate a new array even if this collection is backed by an array). The caller is thus free to modify the returned array.

This method acts as bridge between array-based and collection-based APIs.

Returns:
An array containing all of the elements in this read-only collection.

toArray

`public Object[] **toArray**(Object[] a)`

Returns an array containing all of the elements in this read-only collection.

The runtime type of the returned array is that of the specified array. If the collection fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this collection.

If this collection fits in the specified array with room to spare (i.e., the array has more elements than this collection), the element in the array immediately following the end of the collection is set to `null`. This is useful in determining the length of this collection *only* if the caller knows that this collection does not contain any `null` elements.)

If this collection makes any guarantees as to what order its elements are returned by its iterator, this method must return the elements in the same order.

Like the `toArray` method, this method acts as bridge between array-based and collection-based APIs.

Further, this method allows precise control over the runtime type of the output array, and may, under certain circumstances, be used to save allocation costs

Suppose `l` is a `List` known to contain only strings. The following code can be used to dump the list into a newly allocated array of `String`:

```
String[] x = (String[])l.toArray(new String[0]);
```

Note that `toArray(new Object[0])` is identical in function to `toArray()`.

Parameters:
`a` - the array into which the elements of this collection are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:
An array containing the elements of this read-only collection.

Throws:
`ArrayStoreException` - the runtime type of the specified array is not a super type of the runtime type of every element in this collection.
`NullPointerException` - if the specified array is a `null` reference.

containsAll

`public boolean **containsAll**(Collection c)`

Determines if this read-only collection contains all of the elements in the specified collection.

Parameters:

(continued from last page)

`c` - the collection to be checked for containment in this collection.

Returns:

`true` if this read-only collection contains all of the elements in the specified collection; otherwise `false`.

Throws:

`ClassCastException` - if the types of one or more elements in the specified collection are incompatible with this collection (optional).
`NullPointerException` - if the specified collection contains one or more `null` elements and this collection does not support `null` elements (optional).
`NullPointerException` - if the specified collection is a `null` reference.

See Also:

`contains(Object)`

equals

`public boolean **equals**(Object o)`

Compares the specified object with this read-only collection for equality.

While the `IReadOnlyCollection` interface adds no stipulations to the general contract for the `Object.equals`, programmers who implement the `IReadOnlyCollection` interface "directly" (in other words, create a class that is a `IReadOnlyCollection` but is not `IReadOnlySet` or `IReadOnlyList`) must exercise care if they choose to override the `Object.equals`. It is not necessary to do so, and the simplest course of action is to rely on `Object`'s implementation, but the implementer may wish to implement a "value comparison" in place of the default "reference comparison." (The `IReadOnlySet` and `IReadOnlyList` interface mandate such value comparisons.)

The general contract for the `Object.equals` method states that equals must be symmetric (in other words, `a.equals(b)` if and only if `b.equals(a)`). The contracts for `IReadOnlySet.equals` and `IReadOnlyList.equals` state that lists are only equal to other lists, and sets to other sets. Thus, a custom `equals` method for a collection class that implements neither the `IReadOnlySet` nor `IReadOnlyList` interface must return `false` when this collection is compared to any list or set. (By the same logic, it is not possible to write a class that correctly implements both the `IReadOnlySet` and `IReadOnlyList` interfaces.)

Parameters:

`o` - the `Object` to be compared for equality with this collection.

Returns:

`true` if the specified object is equal to this read-only collection; otherwise `false`.

See Also:

`Object.equals(java.lang.Object)`
`IReadOnlySet.equals(Object)`
`IReadOnlyList.equals(Object)`

hashCode

`public int **hashCode**()`

Returns the hash code value for this read-only collection.

While the `IReadOnlyCollection` interface adds no stipulations to the general contract for the `Object.hashCode` method, programmers should take note that any class that overrides the `Object.equals` method must also override the `Object.hashCode` method in order to satisfy the general contract for the `Object.hashCode`method. In particular, `c1.equals(c2)` implies that `c1.hashCode() == c2.hashCode()`.

Returns:

The hash code value for this read-only collection.

See Also:

`Object.hashCode()`
`Object.equals(java.lang.Object)`

com.imis.util
Interface IReadOnlyList

All Superinterfaces:
IReadOnlyCollection

All Known Implementing Classes:
ReadOnlyArrayList

---

public interface **IReadOnlyList**
extends IReadOnlyCollection

Provides a read-only version of the `List` interface.

A `IReadOnlyList` is simply a `List` without methods that allow changes in the list.
See Also:
`java.util.List`, `ReadOnlyArrayList`

Method Summary

`boolean`	`contains(Object o)` Determines if this read-only list contains the specified element.
`boolean`	`containsAll(Collection c)` Determines if this read-only list contains all of the elements in the specified collection.
`boolean`	`equals(Object o)` Compares the specified object with this read-only list for equality.
`Object`	`get(int index)` Returns the element at the specified position in this read-only list.
`int`	`hashCode()` Returns the hash code value for this read-only list.
`int`	`indexOf(Object o)` Returns the index in this read-only list of the first occurrence of the specified element, or `-1` if this read-only list does not contain this element.
`boolean`	`isEmpty()` Determines if this read-only list contains no elements.
`Iterator`	`iterator()` Returns an iterator over the elements in this read-only list in proper sequence.
`int`	`lastIndexOf(Object o)` Returns the index in this read-only list of the last occurrence of the specified element, or `-1` if this read-only list does not contain this element.
`ListIterator`	`listIterator()` Returns a list iterator of the elements in this read-only list (in proper sequence).
`ListIterator`	`listIterator(int index)` Returns a list iterator of the elements in this read-only list (in proper sequence), starting at the specified position in this read-only list.

`int`	`size()` Returns the number of elements in this read-only list.
`List`	`subList(int fromIndex, int toIndex)` Returns a view of the portion of this read-only list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.
`Object[]`	`toArray()` Returns an array containing all of the elements in this read-only list in proper sequence.
`Object[]`	`toArray(Object[] a)` Returns an array containing all of the elements in this read-only list in proper sequence.

Methods inherited from interface `com.imis.util.IReadOnlyCollection`

`contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `size`, `toArray`, `toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods

size

`public int size()`

Returns the number of elements in this read-only list.

If this list contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.

Returns:
The number of elements in this read-only list.

isEmpty

`public boolean isEmpty()`

Determines if this read-only list contains no elements.

Returns:
`true` if this read-only list contains no elements; otherwise `false`.

contains

`public boolean contains(Object o)`

Determines if this read-only list contains the specified element.

More formally, returns `true` if and only if this list contains at least one element `e` such that `(null == o) ? (null == e) : o.equals(e)`.

Parameters:
`o` - the element whose presence in this list is to be tested.

Returns:
`true` if this read-only list contains the specified element; otherwise `false`.

(continued from last page)

Throws:

`ClassCastException` - if the type of the specified element is incompatible with this list (optional).

`NullPointerException` - if the specified element is a `null` reference and this list does not support `null` elements (optional).

iterator

`public Iterator` **`iterator`**`()`

Returns an iterator over the elements in this read-only list in proper sequence.

Note that `remove` operation is not supported by this iterator.

Returns:

An `Iterator` over the elements in this read-only list in proper sequence.

toArray

`public Object[]` **`toArray`**`()`

Returns an array containing all of the elements in this read-only list in proper sequence.

This method obeys the general contract of the `Collection.toArray` method.

Returns:

An array containing all of the elements in this read-only list in proper sequence.

See Also:

`Arrays#asList(Object[])`

toArray

`public Object[]` **`toArray`**`(Object[] a)`

Returns an array containing all of the elements in this read-only list in proper sequence.

The runtime type of the returned array is that of the specified array.

This method obeys the general contract of the `Collection.toArray(Object[])` method.

Parameters:

`a` - the array into which the elements of this list are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:

An array containing the elements of this read-only list.

Throws:

`ArrayStoreException` - if the runtime type of the specified array is not a super type of the runtime type of every element in this list.
`NullPointerException` - if the specified array is a `null` reference.

containsAll

`public boolean` **`containsAll`**`(Collection c)`

Determines if this read-only list contains all of the elements in the specified collection.

Parameters:

`c` - the collection to be checked for containment in this list.

(continued from last page)

Returns:
`true` if this read-only list contains all of the elements in the specified collection; otherwise `false`.

Throws:
`ClassCastException` - if the types of one or more elements in the specified collection are incompatible with this list (optional).
`NullPointerException` - if the specified collection contains one or more `null` elements and this list does not support `null` elements (optional).
`NullPointerException` - if the specified collection is a `null` reference.

See Also:
`contains(Object)`

equals

`public boolean **equals**(Object o)`

Compares the specified object with this read-only list for equality.

Returns `true` if and only if the specified object is also a list, both lists have the same size, and all corresponding pairs of elements in the two lists are equal. (Two elements `e1` and `e2` are equal if `(null == e1) ? (null == e2) : e1.equals(e2)`.) In other words, two lists are defined to be equal if they contain the same elements in the same order. This definition ensures that the equals method works properly across different implementations of the `List` interface.

Parameters:
`o` - the object to be compared for equality with this list.

Returns:
`true` if the specified object is equal to this read-only list; otherwise `false`.

hashCode

`public int **hashCode**()`

Returns the hash code value for this read-only list.

The hash code of a list is defined to be the result of the following calculation:

```
hashCode = 1;
 Iterator i = list.iterator();
 while (i.hasNext()) {
   Object obj = i.next();
   hashCode = 31 * hashCode + ((null == obj) ? 0 : obj.hashCode());
 }
```

This ensures that `list1.equals(list2)` implies that `list1.hashCode() == list2.hashCode()` for any two lists, `list1` and `list2`, as required by the general contract of `Object.hashCode`.

Returns:
The hash code value for this read-only list.

See Also:
`Object.hashCode()`
`Object.equals(java.lang.Object)`
`equals(Object)`

get

`public Object **get**(int index)`

Returns the element at the specified position in this read-only list.

Parameters:
`index` - the index of element to return.

Returns:
The element at the specified position in this read-only list.

Throws:
`IndexOutOfBoundsException` - if the index is out of range `(0 > index) || (index >= size())`.

indexOf

`public int **indexOf**(Object o)`

Returns the index in this read-only list of the first occurrence of the specified element, or `-1` if this read-only list does not contain this element.

More formally, returns the lowest index `i` such that `(null == o) ? (null == get(i)) : o.equals(get(i)))`, or `-1` if there is no such index.

Parameters:
`o` - the element to search for.

Returns:
The index in this read-only list of the first occurrence of the specified element, or `-1` if this read-only list does not contain this element.

Throws:
`ClassCastException` - if the type of the specified element is incompatible with this list (optional).
`NullPointerException` - if the specified element is a `null` reference and this list does not support `null` elements (optional).

lastIndexOf

`public int **lastIndexOf**(Object o)`

Returns the index in this read-only list of the last occurrence of the specified element, or `-1` if this read-only list does not contain this element.

More formally, returns the highest index `i` such that `(null == o) ? (null == get(i)) : o.equals(get(i)))`, or `-1` if there is no such index.

Parameters:
`o` - the element to search for.

Returns:
The index in this read-only list of the last occurrence of the specified element, or `-1` if this read-only list does not contain this element.

Throws:
`ClassCastException` - if the type of the specified element is incompatible with this list (optional).
`NullPointerException` - if the specified element is a `null` reference and this list does not support `null` elements (optional).

(continued from last page)

listIterator

`public ListIterator **listIterator**()`

Returns a list iterator of the elements in this read-only list (in proper sequence).

Note that modification operations are not supported by this list iterator.

Returns:
A list iterator of the elements in this read-only list (in proper sequence).

listIterator

`public ListIterator **listIterator**(int index)`

Returns a list iterator of the elements in this read-only list (in proper sequence), starting at the specified position in this read-only list.

The specified index indicates the first element that would be returned by an initial call to the `next` method. An initial call to the `previous` method would return the element with the specified index minus one.

Note that modification operations are not supported by this list iterator.

Parameters:
`index` - index of first element to be returned from the list iterator (by a call to the `next` method).

Returns:
A list iterator of the elements in this read-only list (in proper sequence), starting at the specified position in this read-only list.

Throws:
`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index > size()`).

subList

```
public List subList(int fromIndex,
          int toIndex)
```

Returns a view of the portion of this read-only list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive. If `fromIndex` and `toIndex` are equal, the returned list is empty.

The returned list is backed by this read-only list and supports only operations that do not change the list.

Parameters:
`fromIndex` - a low endpoint (inclusive) of the subList.
`toIndex` - a high endpoint (exclusive) of the subList.

Returns:
A view of the specified range within this read-only list.

Throws:
`IndexOutOfBoundsException` - for an illegal endpoint index value (`fromIndex < 0 || toIndex > size || fromIndex > toIndex`).

com.imis.util
Interface IReadOnlyMap

All Known Implementing Classes:
HashBiMap, ReadOnlyTreeMap

public interface **IReadOnlyMap**
extends

Provides a read-only version of the `Map` interface.

A `IReadOnlyMap` is simply a `Map` without methods that allow changes in the map.

The entry set view, like in the `Map`, is a set of `Map.Entry` elements.
See Also:
`java.util.Map`, `ReadOnlyTreeMap`

Method Summary

`boolean`	`containsKey(Object key)` Returns `true` if this read-only map contains a mapping for the specified key.
`boolean`	`containsValue(Object value)` Returns `true` if this read-only map maps one or more keys to the specified value.
`Set`	`entrySet()` Returns a set view of the mappings contained in this read-only map.
`boolean`	`equals(Object o)` Compares the specified object with this read-only map for equality.
`Object`	`get(Object key)` Returns the value to which this read-only map maps the specified key.
`int`	`hashCode()` Returns the hash code value for this read-only map.
`boolean`	`isEmpty()` Checks if this read-only map contains no key-value mappings.
`Set`	`keySet()` Returns a `Set` view of the keys contained in this read-only map.
`int`	`size()` Returns the number of key-value mappings in this read-only map.
`Collection`	`values()` Returns a `Collection` view of the values contained in this read-only map.

Methods

(continued from last page)

size

`public int size()`

Returns the number of key-value mappings in this read-only map.

If the map contains more than `Integer.MAX_VALUE` elements, this method returns `Integer.MAX_VALUE`.

Returns:
The number of key-value mappings in this read-only map.

isEmpty

`public boolean isEmpty()`

Checks if this read-only map contains no key-value mappings.

Returns:
`true` if this read-only map is empty; otherwise `false`.

containsKey

`public boolean containsKey(Object key)`

Returns `true` if this read-only map contains a mapping for the specified key.

More formally, returns `true` if and only if this read-only map contains at a mapping for a key `k` such that `(null == key) ? (null == k) : key.equals(k)`. (There can be at most one such mapping.)

Parameters:
`key` - key whose presence in this map is to be tested.

Returns:
`true` if this map contains a mapping for the specified key; otherwise `false`.

Throws:
`NullPointerException` - if the `key` is `null` and this map does not not permit `null` keys (optional).
`ClassCastException` - if the `key` is of an inappropriate type for this map (optional).

containsValue

`public boolean containsValue(Object value)`

Returns `true` if this read-only map maps one or more keys to the specified value.

More formally, returns `true` if and only if this map contains at least one mapping to a value `v` such that `(null == value) ? (null == v) : value.equals(v)`. This operation will probably require time linear in the map size for most implementations of the `IReadOnlyMap` interface.

Parameters:
`value` - value whose presence in this map is to be tested.

Returns:
`true` if this map maps one or more keys to the specified value; otherwise `false`.

Throws:
`NullPointerException` - if the `value` is `null` and this map does not not permit `null` values (optional).
`ClassCastException` - if the `value` is of an inappropriate type for this map (optional).

get

`public Object get(Object key)`

Returns the value to which this read-only map maps the specified key.

Returns `null` if the map contains no mapping for this key. A return value of `null` does not necessarily indicate that the map contains no mapping for the key; it's also possible that the map explicitly maps the key to `null`. The `containsKey` operation may be used to distinguish these two cases.

More formally, if this map contains a mapping from a key `k` to a value `v` such that `(null == key) ? (null == k) : key.equals(k))`, then this method returns `v`; otherwise it returns `null`. (There can be at most one such mapping.)

Parameters:
`key` - key whose associated value is to be returned.

Returns:
The value to which this map maps the specified key, or a `null` reference if the map contains no mapping for this key.

Throws:
`NullPointerException` - if `key` is `null` and this map does not not permit `null` keys (optional).
`ClassCastException` - if the `key` is of an inappropriate type for this map (optional).

See Also:
`containsKey(Object)`

keySet

`public Set keySet()`

Returns a `Set` view of the keys contained in this read-only map.

The set is backed by the read-only map and does not support modification operations.

Returns:
A `Set` view of the keys contained in this map.

values

`public Collection values()`

Returns a `Collection` view of the values contained in this read-only map.

The collection is backed by the read-only map and does not support modification operations.

Returns:
A `Collection` view of the values contained in this read-only map.

entrySet

`public Set entrySet()`

Returns a set view of the mappings contained in this read-only map.

Each element in this set is a `Map.Entry`.

The set is backed by the read-only map and does not support modification operations.

(continued from last page)

Returns:
A set view of the mappings contained in this read-only map.

equals

`public boolean **equals**(Object o)`

Compares the specified object with this read-only map for equality.

Returns `true` if the given object is also a read-only map and the two read-only maps represent the same mappings. More formally, two read-only maps `t1` and `t2` represent the same mappings if

```
t1.entrySet().equals(t2.entrySet()).
```

This ensures that the `equals` method works properly across different implementations of the `IReadOnlyMap` interface.

Parameters:
`o` - object to be compared for equality with this read-only map.

Returns:
`true` if the specified object is equal to this read-only map; otherwise `false`.

hashCode

`public int **hashCode**()`

Returns the hash code value for this read-only map.

The hash code of a read-only map is defined to be the sum of the hash codes of each entry in the read-only map. This ensures that `t1.equals(t2)` implies that `t1.hashCode() == t2.hashCode()` for any two read-only maps `t1` and `t2`, as required by the general contract of `Object.hashCode()`.

Returns:
The hash code value for this read-only map.

See Also:
`Map.Entry.hashCode()`
`Object.hashCode()`
`Object.equals(java.lang.Object)`
`equals(Object)`

com.imis.util
Interface IReadOnlySet

All Superinterfaces:
IReadOnlyCollection

All Known Implementing Classes:
ReadOnlyTreeSet

---

public interface **IReadOnlySet**
extends IReadOnlyCollection

Provides a read-only version of the `Set` interface.

A `IReadOnlySet` is simply a `Set` without methods that allow changes in the set.
See Also:
`java.util.Set`, `ReadOnlyTreeSet`

Method Summary

`boolean`	`contains(Object o)` Determines if this read-only set contains the specified element.
`boolean`	`containsAll(Collection c)` Determines if this read-only set contains all of the elements in the specified collection.
`boolean`	`equals(Object o)` Compares the specified object with this read-only set for equality.
`int`	`hashCode()` Returns the hash code value for this read-only set.
`boolean`	`isEmpty()` Determines if this read-only set contains no elements.
`Iterator`	`iterator()` Returns an iterator over the elements in this read-only set.
`int`	`size()` Returns the number of elements in this read-only set.
`Object[]`	`toArray()` Returns an array containing all of the elements in this read-only set.
`Object[]`	`toArray(Object[] a)` Returns an array containing all of the elements in this read-only set.

Methods inherited from interface `com.imis.util.IReadOnlyCollection`

`contains, containsAll, equals, hashCode, isEmpty, iterator, size, toArray, toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods

size

`public int size()`

Returns the number of elements in this read-only set.

If this set contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.

Returns:
The number of elements in this read-only set.

isEmpty

`public boolean isEmpty()`

Determines if this read-only set contains no elements.

Returns:
`true` if this read-only set contains no elements; otherwise `false`.

contains

`public boolean contains(Object o)`

Determines if this read-only set contains the specified element.

More formally, returns `true` if and only if this set contains an element `e` such that `(null == o) ? (null == e) : o.equals(e)`.

Parameters:
`o` - the element whose presence in this set is to be tested.

Returns:
`true` if this read-only set contains the specified element; otherwise `false`.

Throws:
`ClassCastException` - if the type of the specified element is incompatible with this set (optional).
`NullPointerException` - if the specified element is a `null` reference and this set does not support `null` elements (optional).

iterator

`public Iterator iterator()`

Returns an iterator over the elements in this read-only set.

The elements are returned in no particular order (unless this set is an instance of some class that provides a guarantee).

Note that `remove` operation is not supported by this iterator.

Returns:
An `Iterator` over the elements in this read-only set.

(continued from last page)

toArray

`public Object[] **toArray**()`

Returns an array containing all of the elements in this read-only set.

Obeys the general contract of the `Collection.toArray` method.

Returns:
An array containing all of the elements in this read-only set.

---

toArray

`public Object[] **toArray**(Object[] a)`

Returns an array containing all of the elements in this read-only set.

The runtime type of the returned array is that of the specified array.

Obeys the general contract of the `Collection.toArray(Object[])` method.

Parameters:
`a` - the array into which the elements of this set are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:
An array containing the elements of this read-only set.

Throws:
`ArrayStoreException` - the runtime type of the specified array is not a super type of the runtime type of every element in this set.
`NullPointerException` - if the specified array is a `null` reference.

---

containsAll

`public boolean **containsAll**(Collection c)`

Determines if this read-only set contains all of the elements in the specified collection.

If the specified collection is also a set, this method returns `true` if it is a subset of this set.

Parameters:
`c` - the collection to be checked for containment in this set.

Returns:
`true` if this read-only set contains all of the elements in the specified collection; otherwise `false`.

Throws:
`ClassCastException` - if the types of one or more elements in the specified collection are incompatible with this set (optional).
`NullPointerException` - if the specified collection contains one or more `null` elements and this collection does not support `null` elements (optional).
`NullPointerException` - if the specified collection is a `null` reference.

See Also:
`contains(Object)`

---

equals

`public boolean **equals**(Object o)`

(continued from last page)

Compares the specified object with this read-only set for equality.

Returns `true` if the specified object is also a set, the two sets have the same size, and every member of the specified set is contained in this set (or equivalently, every member of this set is contained in the specified set). This definition ensures that the equals method works properly across different implementations of the set interface.

Parameters:

`o` - the `Object` to be compared for equality with this set.

Returns:

`true` if the specified object is equal to this read-only set; otherwise `false`.

hashCode

`public int` **`hashCode`**`()`

Returns the hash code value for this read-only set.

The hash code of a set is defined to be the sum of the hash codes of the elements in the set, where the hashcode of a `null` element is defined to be zero. This ensures that `s1.equals(s2)` implies that `s1.hashCode()==s2.hashCode()` for any two sets `s1` and `s2`, as required by the general contract of the `Object.hashCode` method.

Returns:

The hash code value for this read-only set.

See Also:

`Object.hashCode()`
`Object.equals(java.lang.Object)`
`equals(Object)`

com.imis.util
Class Mutex

```
java.lang.Object
  |
  +-com.imis.util.Mutex
```

public class **Mutex**
extends Object

Provides mutual exclusion of threads.

Only one thread can enter critical section guarded by mutex, but can do it several times.

Use the `lock()` and `unlock()` methods to mark the beginning and end of a critical section.
Author:
Robert Petek

Field Summary

`protected`	`mBlocked` The number of threads waiting for the mutex.
`protected`	`mNested` The number of times the owning thread has locked the mutex.
`protected`	`mOwner` Lock owning thread.

Constructor Summary

`public`	`Mutex()` Initializes a new instance of the `Mutex` class.

Method Summary

`void`	`lock()` Locks the mutex.
`boolean`	`lock(long timeout)` Tries to lock the mutex within specified period of time in milliseconds.
`void`	`unlock()` Releases the mutex.

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

(continued from last page)

mOwner

`protected java.lang.Thread **mOwner**`

Lock owning thread.

---

mNested

`protected int **mNested**`

The number of times the owning thread has locked the mutex.

---

mBlocked

`protected int **mBlocked**`

The number of threads waiting for the mutex.

Constructors

Mutex

`public **Mutex**()`

Initializes a new instance of the `Mutex` class.

Methods

lock

`public void **lock**()`

Locks the mutex.

This method should be called before entering critical section.

Throws:
`RuntimeException` - if the current thread has been interrupted.

---

lock

`public boolean **lock**(long timeout)`

Tries to lock the mutex within specified period of time in milliseconds.

This method should be called before entering critical section.

Parameters:
`timeout` - the number of milliseconds to wait.

Returns:
`true` if mutex is successfully locked, `false` if the method was terminated due to timeout expiration.

Throws:
`RuntimeException` - if the current thread has been interrupted.

(continued from last page)

unlock

`public void **unlock**()`

Releases the mutex.

This method should be called after exit from critical section. Mutex will be unlocked only if number of `unlock()` method calls is equal to the number of `lock()` method calls.

Throws:

`IllegalStateException` - if mutex is not locked.

`IllegalMonitorStateException` - if current thread is not owner of the mutex.

com.imis.util
Class Pair

```
java.lang.Object
  |
  +-com.imis.util.Pair
```

All Implemented Interfaces:
Comparable

---

public class **Pair**
extends Object
implements Comparable

Defines a utility class that is used to store a pair of related objects.
Author:
Robert Petek

Constructor Summary

public	`Pair()` Initializes a new instance of the `Pair` class.
public	`Pair(Object first, Object second)` Initializes a new instance of the `Pair` class.

Method Summary

int	`compareTo(Pair pair)` Compares this object with the specified `Pair` object for order.
boolean	`equals(Object obj)` Determines whether the specified `Object` is equal to this instance.
boolean	`equals(Pair obj)` Determines whether the specified `Pair` object is equal to this instance.
Object	`getFirst()` Gets the first object of the object pair.
Object	`getSecond()` Gets the second object of the object pair.
int	`hashCode()` Returns a hash code value for the object.
void	`setFirst(Object obj)` Sets the first object of the object pair.
void	`setSecond(Object obj)` Sets the second object of the object pair.
String	`toString()` Returns the values of the `Pair` members.

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.lang.Comparable`
`compareTo`

Constructors

Pair

```
public Pair()
```

Initializes a new instance of the `Pair` class.

Pair

```
public Pair(Object first,
            Object second)
```

Initializes a new instance of the `Pair` class.

Parameters:
`first` - the first object.
`second` - the second object.

Methods

getFirst

```
public Object getFirst()
```

Gets the first object of the object pair.

Returns:
The first object of the object pair.

setFirst

```
public void setFirst(Object obj)
```

Sets the first object of the object pair.

Parameters:
`obj` - the object to set to the first object.

getSecond

```
public Object getSecond()
```

Gets the second object of the object pair.

Returns:
The second object of the object pair.

setSecond

`public void **setSecond**(Object obj)`

Sets the second object of the object pair.

Parameters:
`obj` - the object to set to the second object.

equals

`public boolean **equals**(Pair obj)`

Determines whether the specified `Pair` object is equal to this instance.

Parameters:
`obj` - the `Pair` object.

Returns:
`true` if the specified `Pair` object is equal to this instance; otherwise, `false`.

equals

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

Determines whether the specified `Object` is equal to this instance.

Parameters:
`obj` - the `java.lang.Object` to compare with this instance.

Returns:
`true` if the specified `Object` is equal to this instance; otherwise, `false`.

hashCode

`public int **hashCode**()`

Returns a hash code value for the object.

This method is supported for the benefit of hashtables such as those provided by `java.util.Hashtable`.

Returns:
A hash code value for this `Pair` object.

compareTo

`public int **compareTo**(Pair pair)`

Compares this object with the specified `Pair` object for order.

Parameters:
`pair` - the `Pair` to be compared.

Returns:
A negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

(continued from last page)

toString

```
public String toString()
```

Returns the values of the `Pair` members.

Returns:
A string containing the values of the `Pair` members.

com.imis.util
Class ReadOnlyArrayList

```
java.lang.Object
  |
  +-java.util.AbstractCollection
      |
      +-java.util.AbstractList
          |
          +-com.imis.util.ReadOnlyArrayList
```

All Implemented Interfaces:
Serializable, Cloneable, RandomAccess, List, IReadOnlyList, Collection, List

---

public class **ReadOnlyArrayList**
extends AbstractList
implements List, Collection, IReadOnlyList, List, RandomAccess, Cloneable, Serializable

Provides an implementation `IReadOnlyList` and `List` interfaces.

A `ReadOnlyArrayList` is simply an `ArrayList` wrapper that prevents modifying the list.
See Also:
`IReadOnlyList`, `java.util.List`, `java.util.ArrayList`

Fields inherited from class `java.util.AbstractList`
`modCount`

Constructor Summary

`public`	`ReadOnlyArrayList()` Initializes a new instance of the `ReadOnlyArrayList` class that wraps an empty array list.
`public`	`ReadOnlyArrayList(IReadOnlyList c)` Initializes a new instance of the `ReadOnlyArrayList` class that wraps the supplied `IReadOnlyList`.
`public`	`ReadOnlyArrayList(Collection c)` Initializes a new instance of the `ReadOnlyArrayList` class that wraps the supplied `java.util.Collection`.

Method Summary

`Object`	`clone()` Returns a shallow copy of this `ReadOnlyArrayList` instance.
`boolean`	`contains(Object elem)` Returns `true` if this read-only list contains the specified element.
`Object`	`get(int index)` Returns the element at the specified position in this read-only list.
`int`	`hashCode()` Returns the hash code value for this read-only list.

int	`indexOf(Object elem)` Searches for the first occurrence of the given argument, testing for equality using the `equals` method.
boolean	`isEmpty()` Returns `true` if this read-only list contains no elements.
int	`lastIndexOf(Object elem)` Returns the index of the last occurrence of the specified object in this read-only list.
int	`size()` Returns the number of elements in this read-only list.
Object[]	`toArray()` Returns an array containing all of the elements in this read-only list in the correct order.
Object[]	`toArray(Object[] a)` Returns an array containing all of the elements in this read-only list in the correct order; the runtime type of the returned array is that of the specified array.

Methods inherited from class `java.util.AbstractList`

`add, add, addAll, clear, equals, get, hashCode, indexOf, iterator, lastIndexOf, listIterator, listIterator, remove, removeRange, set, subList`

Methods inherited from class `java.util.AbstractCollection`

`add, addAll, clear, contains, containsAll, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.util.Collection`

`add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods inherited from interface `java.util.List`

`add, add, addAll, addAll, clear, contains, containsAll, equals, get, hashCode, indexOf, isEmpty, iterator, lastIndexOf, listIterator, listIterator, remove, remove, removeAll, retainAll, set, size, subList, toArray, toArray`

Methods inherited from interface `java.util.Collection`

`add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods inherited from interface `com.imis.util.IReadOnlyList`
`contains`, `containsAll`, `equals`, `get`, `hashCode`, `indexOf`, `isEmpty`, `iterator`, `lastIndexOf`, `listIterator`, `listIterator`, `size`, `subList`, `toArray`, `toArray`

Methods inherited from interface `com.imis.util.IReadOnlyCollection`
`contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `size`, `toArray`, `toArray`

Methods inherited from interface `java.lang.Iterable`
`iterator`

Methods inherited from interface `java.util.List`
`add`, `add`, `addAll`, `addAll`, `clear`, `contains`, `containsAll`, `equals`, `get`, `hashCode`, `indexOf`, `isEmpty`, `iterator`, `lastIndexOf`, `listIterator`, `listIterator`, `remove`, `remove`, `removeAll`, `retainAll`, `set`, `size`, `subList`, `toArray`, `toArray`

Methods inherited from interface `java.util.Collection`
`add`, `addAll`, `clear`, `contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `remove`, `removeAll`, `retainAll`, `size`, `toArray`, `toArray`

Methods inherited from interface `java.lang.Iterable`
`iterator`

Constructors

ReadOnlyArrayList

`public **ReadOnlyArrayList**()`

Initializes a new instance of the `ReadOnlyArrayList` class that wraps an empty array list.

ReadOnlyArrayList

`public **ReadOnlyArrayList**(IReadOnlyList c)`

Initializes a new instance of the `ReadOnlyArrayList` class that wraps the supplied `IReadOnlyList`.

Parameters:
`c` - the read-only list to wrap.

Throws:
`NullPointerException` - if `c` is a `null` reference.

ReadOnlyArrayList

`public **ReadOnlyArrayList**(Collection c)`

Initializes a new instance of the `ReadOnlyArrayList` class that wraps the supplied `java.util.Collection`.

Parameters:

(continued from last page)

`c` - the collection whose elements are to be placed in this read-only list.

Throws:
`NullPointerException` - if `c` is a `null` reference.

Methods

size

`public int size()`

Returns the number of elements in this read-only list.

Returns:
The number of elements in this read-only list.

---

isEmpty

`public boolean isEmpty()`

Returns `true` if this read-only list contains no elements.

Returns:
`true` if this read-only list has no elements; otherwise `false`.

---

contains

`public boolean contains(Object elem)`

Returns `true` if this read-only list contains the specified element.

Parameters:
`elem` - element whose presence in this list is to be tested.

Returns:
`true` if the specified element is present; otherwise `false`.

---

get

`public Object get(int index)`

Returns the element at the specified position in this read-only list.

Parameters:
`index` - the index of element to return.

Returns:
The element at the specified position in this read-only list.

Throws:
`IndexOutOfBoundsException` - if index is out of range `(index < 0) || (index >= size())`.

---

indexOf

`public int indexOf(Object elem)`

Searches for the first occurrence of the given argument, testing for equality using the `equals` method.

(continued from last page)

Parameters:
`elem` - an object.

Returns:
The index of the first occurrence of the argument in this list; returns `-1` if the object is not found.

See Also:
`Object.equals(java.lang.Object)`

lastIndexOf

`public int lastIndexOf(Object elem)`

Returns the index of the last occurrence of the specified object in this read-only list.

Parameters:
`elem` - the desired element.

Returns:
The index of the last occurrence of the specified object in this list; returns `-1` if the object is not found.

clone

`public Object clone()`

Returns a shallow copy of this `ReadOnlyArrayList` instance.

The elements themselves are not copied.

Returns:
A clone of this read-only list instance.

hashCode

`public int hashCode()`

Returns the hash code value for this read-only list.

Returns:
The hash code value for this read-only list.

toArray

`public Object[] toArray()`

Returns an array containing all of the elements in this read-only list in the correct order.

Returns:
An array containing all of the elements in this read-only list in the correct order.

toArray

`public Object[] toArray(Object[] a)`

(continued from last page)

Returns an array containing all of the elements in this read-only list in the correct order; the runtime type of the returned array is that of the specified array.

If the list fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this list.

If the list fits in the specified array with room to spare (i.e., the array has more elements than the list), the element in the array immediately following the end of the collection is set to `null`. This is useful in determining the length of the list only if the caller knows that the list does not contain any `null` elements.

Parameters:

`a` - the array into which the elements of the list are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:

An array containing the elements of the read-only list.

Throws:

`ArrayStoreException` - if the runtime type of a is not a super type of the runtime type of every element in this list.

com.imis.util
Class ReadOnlyTreeMap

```
java.lang.Object
  |
  +-java.util.AbstractMap
      |
      +-com.imis.util.ReadOnlyTreeMap
```

All Implemented Interfaces:
Serializable, Cloneable, SortedMap, IReadOnlyMap, Map

public class **ReadOnlyTreeMap**
extends AbstractMap
implements Map, IReadOnlyMap, SortedMap, Cloneable, Serializable

Provides an implementation `IReadOnlyMap` and `SortedMap` interfaces.

A `ReadOnlyTreeMap` is simply a `TreeMap` wrapper that prevents modifying the map.
See Also:
`IReadOnlyMap`, `java.util.SortedMap`, `java.util.TreeMap`

Constructor Summary

`public`	`ReadOnlyTreeMap()` Initializes a new instance of the `ReadOnlyTreeMap` class that wraps an empty tree map.
`public`	`ReadOnlyTreeMap(IReadOnlyMap m)` Initializes a new instance of the `ReadOnlyTreeMap` class that wraps the supplied `IReadOnlyMap`.
`public`	`ReadOnlyTreeMap(Map m)` Initializes a new instance of the `ReadOnlyTreeMap` class that wraps the supplied `java.util.Map`.
`public`	`ReadOnlyTreeMap(SortedMap m)` Initializes a new instance of the `ReadOnlyTreeMap` class that wraps the supplied `java.util.SortedMap`.

Method Summary

`Object`	`clone()` Returns a shallow copy of this `ReadOnlyTreeMap` instance.
`Comparator`	`comparator()` Returns the comparator used to order this read-only map, or `null` if this map uses its keys natural order.
`boolean`	`containsKey(Object key)` Returns `true` if this read-only map contains a mapping for the specified key.
`boolean`	`containsValue(Object value)` Returns `true` if this read-only map maps one or more keys to this value.
`Set`	`entrySet()` Returns a `Set` view of the mappings contained in this read-only map.

`Object`	`firstKey()` Returns the first (lowest) key currently in this sorted read-only map.
`Object`	`get(Object key)` Returns the value to which this map maps the specified key or a `null` reference, if the map contains no mapping for this key.
`int`	`hashCode()` Returns the hash code value for this read-only map.
`SortedMap`	`headMap(Object toKey)` Returns a view of the portion of this read-only map whose keys are strictly less than `toKey`.
`boolean`	`isEmpty()` Returns `true` if this read-only map contains no key-value mappings.
`Object`	`lastKey()` Returns the last (highest) key currently in this sorted read-only map.
`int`	`size()` Returns the number of key-value mappings in this read-only map.
`SortedMap`	`subMap(Object fromKey, Object toKey)` Returns a view of the portion of this read-only map whose keys range from `fromKey`, inclusive, to `toKey`, exclusive.
`SortedMap`	`tailMap(Object fromKey)` Returns a view of the portion of this read-only map whose keys are greater than or equal to `fromKey`.

Methods inherited from class `java.util.AbstractMap`

`clear, clone, containsKey, containsValue, entrySet, equals, get, hashCode, isEmpty, keySet, put, putAll, remove, size, toString, values`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.util.Map`

`clear, containsKey, containsValue, entrySet, equals, get, hashCode, isEmpty, keySet, put, putAll, remove, size, values`

Methods inherited from interface `com.imis.util.IReadOnlyMap`

`containsKey, containsValue, entrySet, equals, get, hashCode, isEmpty, keySet, size, values`

Methods inherited from interface `java.util.SortedMap`

`comparator, firstKey, headMap, lastKey, subMap, tailMap`

Methods inherited from interface `java.util.Map`

`clear, containsKey, containsValue, entrySet, equals, get, hashCode, isEmpty, keySet, put, putAll, remove, size, values`

Constructors

ReadOnlyTreeMap

`public` **`ReadOnlyTreeMap`**`()`

Initializes a new instance of the `ReadOnlyTreeMap` class that wraps an empty tree map.

ReadOnlyTreeMap

`public` **`ReadOnlyTreeMap`**`(IReadOnlyMap m)`

Initializes a new instance of the `ReadOnlyTreeMap` class that wraps the supplied `IReadOnlyMap`.

Parameters:
`m` - the read-only map to wrap.

Throws:
`NullPointerException` - if `m` is a `null` reference or its comparator does not tolerate `null` keys.

ReadOnlyTreeMap

`public` **`ReadOnlyTreeMap`**`(Map m)`

Initializes a new instance of the `ReadOnlyTreeMap` class that wraps the supplied `java.util.Map`.

Parameters:
`m` - the map whose mappings are to be placed in this read-only map.

Throws:
`NullPointerException` - if `m` is a `null` reference or this read-only map does not permit `null` keys and a key in the specified map is `null`.
`ClassCastException` - class of a key or value in the specified map prevents it from being stored in this map.

ReadOnlyTreeMap

`public` **`ReadOnlyTreeMap`**`(SortedMap m)`

Initializes a new instance of the `ReadOnlyTreeMap` class that wraps the supplied `java.util.SortedMap`.

Parameters:
`m` - the sorted map whose mappings are to be placed in this map, and whose comparator is to be used to sort this map.

Throws:
`NullPointerException` - if `m` is a `null` reference or this read-only map does not permit `null` keys and a key in the specified map is `null`.
`ClassCastException` - class of a key or value in the specified map prevents it from being stored in this map.

Methods

size

`public int` **`size`**`()`

Returns the number of key-value mappings in this read-only map.

(continued from last page)

Returns:
the number of key-value mappings in this read-only map.

isEmpty

`public boolean **isEmpty**()`

Returns `true` if this read-only map contains no key-value mappings.

Returns:
`true` if this read-only map contains no key-value mappings; otherwise `false`.

comparator

`public Comparator **comparator**()`

Returns the comparator used to order this read-only map, or `null` if this map uses its keys natural order.

Returns:
The comparator used to order this read-only map, or `null` if this map uses its keys natural order.

containsKey

`public boolean **containsKey**(Object key)`

Returns `true` if this read-only map contains a mapping for the specified key.

Parameters:
`key` - the key whose presence in this map is to be tested.

Returns:
`true` if this read-only map contains a mapping for the specified key; otherwise `false`.

Throws:
`NullPointerException` - if `key` is a `null` reference and this map does not not permit `null` keys.

containsValue

`public boolean **containsValue**(Object value)`

Returns `true` if this read-only map maps one or more keys to this value.

Parameters:
`value` - the value whose presence in this map is to be tested.

Returns:
`true` if this read-only map maps one or more keys to this value; otherwise `false`.

get

`public Object **get**(Object key)`

Returns the value to which this map maps the specified key or a `null` reference, if the map contains no mapping for this key.

A return value of `null` does not *necessarily* indicate that the map contains no mapping for the key; it's also possible that the map explicitly maps the key to `null`. The `containsKey()` operation may be used to distinguish these two cases.

Parameters:

(continued from last page)

`key` - the key whose associated value is to be returned.

Returns:
The value to which this read-only map maps the specified key.

Throws:
`NullPointerException` - if the key is a `null` reference and this map does not not permit `null` keys.

See Also:
`containsKey(Object)`

clone

`public Object **clone**()`

Returns a shallow copy of this `ReadOnlyTreeMap` instance.

The keys and values themselves are not cloned.

Returns:
A shallow copy of this read-only map.

entrySet

`public Set **entrySet**()`

Returns a `Set` view of the mappings contained in this read-only map.

The set's iterator returns the mappings in ascending key order. Each element in this set is a `Map.Entry`. The set is backed by this read-only map and supports only operations that do not change the set.

Returns:
A read-only set view of the mappings contained in this read-only map.

hashCode

`public int **hashCode**()`

Returns the hash code value for this read-only map.

The hash code for this read-only map is equal to the hash code of the underlying `TreeMap`.

Returns:
The hash code value for this read-only map.

See Also:
`AbstractMap.hashCode()`

subMap

```
public SortedMap subMap(Object fromKey,
        Object toKey)
```

Returns a view of the portion of this read-only map whose keys range from `fromKey`, inclusive, to `toKey`, exclusive. If `fromKey` and `toKey` are equal, the returned sorted map is empty.

The returned sorted map is backed by this read-only map and supports only operations that do not change the map.

Parameters:
`fromKey` - low endpoint (inclusive) of the subMap.

(continued from last page)

`toKey` - high endpoint (exclusive) of the subMap.

Returns:

A view of the portion of this read-only map whose keys range from `fromKey`, inclusive, to `toKey`, exclusive.

Throws:

`ClassCastException` - if `fromKey` and `toKey` cannot be compared to one another using this read-only map's comparator (or, if the map has no comparator, using natural ordering).
`IllegalArgumentException` - if `fromKey` is greater than `toKey`.
`NullPointerException` - if `fromKey` or `toKey` is `null` and this read-only map uses natural order, or its comparator does not tolerate `null` keys.

headMap

`public SortedMap **headMap**(Object toKey)`

Returns a view of the portion of this read-only map whose keys are strictly less than `toKey`.

The returned sorted map is backed by this read-only map and supports only operations that do not change the map.

Parameters:

`toKey` - high endpoint (exclusive) of the headMap.

Returns:

A view of the portion of this read-only map whose keys are strictly less than `toKey`.

Throws:

`ClassCastException` - if `toKey` is not compatible with this read-only map's comparator (or, if the map has no comparator, if `toKey` does not implement `Comparable`).
`IllegalArgumentException` - if this read-only map is itself a subMap, headMap, or tailMap, and `toKey` is not within the specified range of the subMap, headMap, or tailMap.
`NullPointerException` - if `toKey` is `null` and this read-only map uses natural order, or its comparator does not tolerate `null` keys.

tailMap

`public SortedMap **tailMap**(Object fromKey)`

Returns a view of the portion of this read-only map whose keys are greater than or equal to `fromKey`.

The returned sorted map is backed by this read-only map and supports only operations that do not change the map.

Parameters:

`fromKey` - low endpoint (inclusive) of the tailMap.

Returns:

A view of the portion of this read-only map whose keys are greater than or equal to `fromKey`.

Throws:

`ClassCastException` - if `fromKey` is not compatible with this map's comparator (or, if the map has no comparator, if `fromKey` does not implement `Comparable`).
`IllegalArgumentException` - if this map is itself a subMap, headMap, or tailMap, and `fromKey` is not within the specified range of the subMap, headMap, or tailMap.
`NullPointerException` - if `fromKey` is `null` and this read-only map uses natural order, or its comparator does not tolerate `null` keys.

firstKey

`public Object **firstKey**()`

Returns the first (lowest) key currently in this sorted read-only map.

(continued from last page)

Returns:
The first (lowest) key currently in this sorted read-only map.

Throws:
`NoSuchElementException` - if read-only map is empty.

lastKey

```
public Object lastKey()
```

Returns the last (highest) key currently in this sorted read-only map.

Returns:
The last (highest) key currently in this sorted read-only map.

Throws:
`NoSuchElementException` - if read-only map is empty.

com.imis.util
Class ReadOnlyTreeSet

```
java.lang.Object
  |
  +-java.util.AbstractCollection
      |
      +-java.util.AbstractSet
          |
          +-com.imis.util.ReadOnlyTreeSet
```

All Implemented Interfaces:
Serializable, Cloneable, SortedSet, IReadOnlySet, Collection, Set

---

public class **ReadOnlyTreeSet**
extends AbstractSet
implements Set, Collection, IReadOnlySet, SortedSet, Cloneable, Serializable

Provides an implementation `IReadOnlySet` and `SortedSet` interfaces.

A `ReadOnlyTreeSet` is simply a `TreeSet` wrapper that prevents modifying the set.
See Also:
`IReadOnlySet`, `java.util.SortedSet`, `java.util.TreeSet`

Constructor Summary

`public`	`ReadOnlyTreeSet()` Initializes a new instance of the `ReadOnlyTreeSet` class that wraps an empty tree set.
`public`	`ReadOnlyTreeSet(IReadOnlySet s)` Initializes a new instance of the `ReadOnlyTreeSet` class that wraps the supplied `IReadOnlySet`.
`public`	`ReadOnlyTreeSet(Collection c)` Initializes a new instance of the `ReadOnlyTreeSet` class that wraps the supplied `java.util.Collection`.
`public`	`ReadOnlyTreeSet(SortedSet s)` Initializes a new instance of the `ReadOnlyTreeSet` class that wraps the supplied `IReadOnlySet`.

Method Summary

`Object`	`clone()` Returns a shallow copy of this `ReadOnlyTreeSet` instance.
`Comparator`	`comparator()` Returns the comparator used to order this read-only sorted set or a `null` reference if this tree set uses its elements natural ordering.
`boolean`	`contains(Object o)` Returns `true` if this read-only set contains the specified element.
`Object`	`first()` Returns the first (lowest) element currently in this read-only sorted set.

`SortedSet`	`headSet(Object toElement)` Returns a view of the portion of this read-only set whose elements are strictly less than `toElement`.
`boolean`	`isEmpty()` Returns `true` if this read-only set contains no elements.
`Iterator`	`iterator()` Returns an iterator over the elements in this read-only set.
`Object`	`last()` Returns the last (highest) element currently in this read-only sorted set.
`int`	`size()` Returns the number of elements in this read-only set (its cardinality).
`SortedSet`	`subSet(Object fromElement, Object toElement)` Returns a view of the portion of this read-only set whose elements range from `fromElement`, inclusive, to `toElement`, exclusive.
`SortedSet`	`tailSet(Object fromElement)` Returns a view of the portion of this read-only set whose elements are greater than or equal to `fromElement`.

Methods inherited from class `java.util.AbstractSet`

`equals, hashCode, removeAll`

Methods inherited from class `java.util.AbstractCollection`

`add, addAll, clear, contains, containsAll, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.util.Collection`

`add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods inherited from interface `java.util.Set`

`add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray`

Methods inherited from interface `java.util.Collection`

`add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods inherited from interface `com.imis.util.IReadOnlySet`

`contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `size`, `toArray`, `toArray`

Methods inherited from interface `com.imis.util.IReadOnlyCollection`

`contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `size`, `toArray`, `toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Methods inherited from interface `java.util.SortedSet`

`comparator`, `first`, `headSet`, `last`, `subSet`, `tailSet`

Methods inherited from interface `java.util.Set`

`add`, `addAll`, `clear`, `contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `remove`, `removeAll`, `retainAll`, `size`, `toArray`, `toArray`

Methods inherited from interface `java.util.Collection`

`add`, `addAll`, `clear`, `contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `remove`, `removeAll`, `retainAll`, `size`, `toArray`, `toArray`

Methods inherited from interface `java.lang.Iterable`

`iterator`

Constructors

ReadOnlyTreeSet

`public ReadOnlyTreeSet()`

Initializes a new instance of the `ReadOnlyTreeSet` class that wraps an empty tree set.

ReadOnlyTreeSet

`public ReadOnlyTreeSet(IReadOnlySet s)`

Initializes a new instance of the `ReadOnlyTreeSet` class that wraps the supplied `IReadOnlySet`.

Parameters:

`s` - the read-only set to wrap.

Throws:

`NullPointerException` - if `s` is a `null` reference.

(continued from last page)

ReadOnlyTreeSet

`public ReadOnlyTreeSet(Collection c)`

Initializes a new instance of the `ReadOnlyTreeSet` class that wraps the supplied `java.util.Collection`.

Parameters:
`c` - the elements that will comprise the new read-only set.

Throws:
`NullPointerException` - if `c` is a `null` reference.

ReadOnlyTreeSet

`public ReadOnlyTreeSet(SortedSet s)`

Initializes a new instance of the `ReadOnlyTreeSet` class that wraps the supplied `IReadOnlySet`.

Parameters:
`s` - the sorted set whose elements will comprise the new read-only set.

Throws:
`NullPointerException` - if `s` is a `null` reference.

Methods

size

`public int size()`

Returns the number of elements in this read-only set (its cardinality).

Returns:
the number of elements in this read-only set (its cardinality).

isEmpty

`public boolean isEmpty()`

Returns `true` if this read-only set contains no elements.

Returns:
`true` if this read-only set contains no elements; otherwise `false`.

comparator

`public Comparator comparator()`

Returns the comparator used to order this read-only sorted set or a `null` reference if this tree set uses its elements natural ordering.

Returns:
The comparator used to order this read-only sorted set or a `null` reference if this tree set uses its elements natural ordering.

iterator

`public Iterator iterator()`

(continued from last page)

Returns an iterator over the elements in this read-only set.

The elements are returned in ascending order.

Returns:
An iterator over the elements in this read-only set.

contains

```
public boolean contains(Object o)
```

Returns `true` if this read-only set contains the specified element.

Parameters:
`o` - the object to be checked for containment in this set.

Returns:
`true` if this read-only set contains the specified element; otherwise `false`.

Throws:
`ClassCastException` - if the specified object cannot be compared with the elements currently in the set.

clone

```
public Object clone()
```

Returns a shallow copy of this `ReadOnlyTreeSet` instance.

The elements themselves are not cloned.

Returns:
A shallow copy of this read-only set.

subSet

```
public SortedSet subSet(Object fromElement,
         Object toElement)
```

Returns a view of the portion of this read-only set whose elements range from `fromElement`, inclusive, to `toElement`, exclusive. If `fromElement` and `toElement` are equal, the returned sorted set is empty.

The returned sorted set is backed by this read-only set and supports only operations that do not change the set.

Parameters:
`fromElement` - the low endpoint (inclusive) of the subSet.
`toElement` - the high endpoint (exclusive) of the subSet.

Returns:
A view of the portion of this read-only set whose elements range from `fromElement`, inclusive, to `toElement`, exclusive.

Throws:
`ClassCastException` - if `fromElement` and `toElement` cannot be compared to one another using this set's comparator (or, if the set has no comparator, using natural ordering).
`IllegalArgumentException` - if `fromElement` is greater than `toElement`.
`NullPointerException` - if `fromElement` or `toElement` is a `null` reference and this set uses natural order, or its comparator does not tolerate `null` elements.

headSet

```
public SortedSet headSet(Object toElement)
```

(continued from last page)

Returns a view of the portion of this read-only set whose elements are strictly less than `toElement`.

The returned sorted set is backed by this read-only set and supports only operations that do not change the set.

Parameters:
`toElement` - the high endpoint (exclusive) of the headSet.

Returns:
A view of the portion of this read-only set whose elements are strictly less than `toElement`.

Throws:
`ClassCastException` - if `toElement` is not compatible with this set's comparator (or, if the set has no comparator, if `toElement` does not implement `Comparable`).
`IllegalArgumentException` - if this set is itself a subSet, headSet, or tailSet, and `toElement` is not within the specified range of the subSet, headSet, or tailSet.
`NullPointerException` - if `toElement` is a `null` reference and this set uses natural ordering, or its comparator does not tolerate `null` elements.

tailSet

`public SortedSet **tailSet**(Object fromElement)`

Returns a view of the portion of this read-only set whose elements are greater than or equal to `fromElement`.

The returned sorted set is backed by this read-only set and supports only operations that do not change the set.

Parameters:
`fromElement` - the low endpoint (inclusive) of the tailSet.

Returns:
A view of the portion of this read-only set whose elements are greater than or equal to `fromElement`.

Throws:
`ClassCastException` - if `fromElement` is not compatible with this set's comparator (or, if the set has no comparator, if `fromElement` does not implement `Comparable`).
`IllegalArgumentException` - if this set is itself a subSet, headSet, or tailSet, and `fromElement` is not within the specified range of the subSet, headSet, or tailSet.
`NullPointerException` - if `fromElement` is a `null` reference and this set uses natural ordering, or its comparator does not tolerate `null` elements.

first

`public Object **first**()`

Returns the first (lowest) element currently in this read-only sorted set.

Returns:
the first (lowest) element currently in this read-only sorted set.

Throws:
`NoSuchElementException` - sorted set is empty.

last

`public Object **last**()`

Returns the last (highest) element currently in this read-only sorted set.

Returns:
the last (highest) element currently in this read-only sorted set.

(continued from last page)

Throws:

`NoSuchElementException` - sorted set is empty.

com.imis.util
Class XmlConvert

```
java.lang.Object
  |
  +-com.imis.util.XmlConvert
```

public class **XmlConvert**
extends Object

Encodes and decodes XML names and provides methods for converting between runtime types and XML Schema definition language (XSD) types.

Constructor Summary

public	`XmlConvert()`

Method Summary

static String	`decodeName(String name)` Decodes a name.
static String	`encodeName(String name)` Converts the name to a valid XML name.
static boolean	`isPublicId(String str)` Checks the specified string if all the characters in the string are valid public id characters.
static boolean	`isPublicIdChar(int ch)` Checks if the specified character is a valid public id character.
static boolean	`isWhitespace(String str)` Checks if all characters in the specified string are valid XML whitespace characters.
static boolean	`isWhitespaceChar(int ch)` Checks if the specified character is a valid XML whitespace character.
static java.math.BigDecimal	`toBigDecimal(String s)` Converts the `String` to a `BigDecimal` equivalent.
static java.math.BigInteger	`toBigInteger(String s)` Converts the `String` to a `BigInteger` equivalent.
static boolean	`toBoolean(String s)` Converts the `String` to a boolean equivalent.
static byte	`toByte(String s)` Converts the `String` representing signed byte value to a byte equivalent.
static Calendar	`toCalendar(String s)` Converts the `String` to a `Calendar` equivalent.
static char	`toChar(String s)` Converts the `String` to a char equivalent.

`static double`	`toDouble(String s)` Converts the `String` to a double equivalent.
`static float`	`toFloat(String s)` Converts the `String` to a float equivalent.
`static int`	`toInt(String s)` Converts the `String` representing signed int value to a int equivalent.
`static long`	`toLong(String s)` Converts the `String` representing signed long value to a long equivalent.
`static String`	`toPattern(Calendar calendar, boolean includeTimeZone)` Creates a pattern describing the date and time format from a given `Calendar` value.
`static short`	`toShort(String s)` Converts the `String` representing signed short value to a short equivalent.
`static String`	`toString(java.math.BigDecimal value)` Converts the `BigDecimal` to a `String`.
`static String`	`toString(java.math.BigInteger value)` Converts the `BigInteger` to a `String`.
`static String`	`toString(boolean b)` Converts the boolean to a `String`.
`static String`	`toString(byte b)` Converts the byte representing signed byte value to a `String`.
`static String`	`toString(byte b, boolean unsigned)` Converts the byte representing signed or unsigned byte value to a `String`.
`static String`	`toString(Calendar calendar)` Converts the `Calendar` to a `String`.
`static String`	`toString(Calendar calendar, String pattern)` Converts the `Calendar` to a `String` using a specified pattern describing the date and time format.
`static String`	`toString(char ch)` Converts the char to a `String`.
`static String`	`toString(double d)` Converts the double to a `String`.
`static String`	`toString(float f)` Converts the float to a `String`.
`static String`	`toString(int i)` Converts the int representing signed int value to a `String`.
`static String`	`toString(int i, boolean unsigned)` Converts the int representing signed or unsigned int value to a `String`.
`static String`	`toString(long l)` Converts the long representing signed long value to a `String`.
`static String`	`toString(long l, boolean unsigned)` Converts the long representing signed or unsigned long value to a `String`.

`static String`	`toString(short s)` Converts the short representing signed short value to a `String`.
`static String`	`toString(short s, boolean unsigned)` Converts the short representing signed or unsigned short value to a `String`.
`static byte`	`toUByte(String s)` Converts the `String` representing unsigned byte value to a byte equivalent.
`static int`	`toUInt(String s)` Converts the `String` representing unsigned int value to a int equivalent.
`static long`	`toULong(String s)` Converts the `String` representing unsigned long value to a long equivalent.
`static short`	`toUShort(String s)` Converts the `String` representing unsigned short value to a short equivalent.

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

XmlConvert

`public XmlConvert()`

Methods

isWhitespaceChar

`public static boolean isWhitespaceChar(int ch)`

Checks if the specified character is a valid XML whitespace character.

Parameters:
`ch` - the character to validate.

Returns:
`true` if the specified character is a valid XML whitespace character; otherwise `false`.

---

isWhitespace

`public static boolean isWhitespace(String str)`

Checks if all characters in the specified string are valid XML whitespace characters.

Parameters:
`str` - the string to validate.

Returns:
`true` if all characters in the specified string is a valid XML whitespace character; otherwise `false`.

isPublicIdChar

`public static boolean **isPublicIdChar**(int ch)`

Checks if the specified character is a valid public id character.

Parameters:
`ch` - the character to verify as a public id character.

Returns:
`true` if the character is a valid public id character; otherwise, `false`.

isPublicId

`public static boolean **isPublicId**(String str)`

Checks the specified string if all the characters in the string are valid public id characters.

Parameters:
`str` - the string that contains the id to validate.

Returns:
`true` if all the characters in the string are valid public id characters; otherwise, `false`.

encodeName

`public static String **encodeName**(String name)`

Converts the name to a valid XML name.

Any XML name character that does not conform to the W3C Extensible Markup Language (XML) 1.0 specification is escaped as "_xHHHH_", where "HHHH" string stands for the four-digit hexadecimal UCS-2 code for the character in most significant bit first order.

Parameters:
`name` - the name to be translated.

Returns:
The name with any invalid characters replaced by an escape string.

decodeName

`public static String **decodeName**(String name)`

Decodes a name.

This method does the reverse of the `encodeName(String)` method.

The names are decoded from left to right. Any sequence "_xHHHH_" (where HHHH stands for a valid, four digit hexadecimal UCS-2 code) that has not been decoded is transformed into the corresponding Unicode character. Short forms are not recognized and are passed on without translation.

Parameters:
`name` - the name to be decoded.

Returns:
The decoded name.

(continued from last page)

toString

```
public static String toString(boolean b)
```

Converts the boolean to a `String`.

Parameters:
`b` - the value to convert.

Returns:
A string representation of the boolean, that is, "true" or "false".

toString

```
public static String toString(char ch)
```

Converts the char to a `String`.

Parameters:
`ch` - the value to convert.

Returns:
A string representation of the char.

toString

```
public static String toString(byte b)
```

Converts the byte representing signed byte value to a `String`.

Parameters:
`b` - the value to convert.

Returns:
A string representation of the byte.

toString

```
public static String toString(byte b,
         boolean unsigned)
```

Converts the byte representing signed or unsigned byte value to a `String`.

Parameters:
`b` - the value to convert.
`unsigned` - `true` if `b` represents an unsigned byte value; otherwise `false`.

Returns:
A string representation of the byte.

toString

```
public static String toString(short s)
```

Converts the short representing signed short value to a `String`.

Parameters:
`s` - the value to convert.

(continued from last page)

Returns:
A string representation of the short.

toString

```
public static String toString(short s,
         boolean unsigned)
```

Converts the short representing signed or unsigned short value to a `String`.

Parameters:
`s` - the value to convert.
`unsigned` - `true` if `s` represents an unsigned short value; otherwise `false`.

Returns:
A string representation of the short.

toString

```
public static String toString(int i)
```

Converts the int representing signed int value to a `String`.

Parameters:
`i` - the value to convert.

Returns:
A string representation of the int.

toString

```
public static String toString(int i,
         boolean unsigned)
```

Converts the int representing signed or unsigned int value to a `String`.

Parameters:
`i` - the value to convert.
`unsigned` - `true` if `i` represents an unsigned int value; otherwise `false`.

Returns:
A string representation of the int.

toString

```
public static String toString(long l)
```

Converts the long representing signed long value to a `String`.

Parameters:
`l` - the value to convert.

Returns:
A string representation of the long.

(continued from last page)

toString

```
public static String toString(long l,
         boolean unsigned)
```

Converts the long representing signed or unsigned long value to a `String`.

Parameters:
`l` - the value to convert.
`unsigned` - `true` if `l` represents an unsigned long value; otherwise `false`.

Returns:
A string representation of the long.

toString

```
public static String toString(java.math.BigInteger value)
```

Converts the `BigInteger` to a `String`.

Parameters:
`value` - the value to convert.

Returns:
A string representation of the `BigInteger`.

Throws:
`NullPointerException` - if `value` is a `null` reference.

toString

```
public static String toString(float f)
```

Converts the float to a `String`.

Parameters:
`f` - the value to convert.

Returns:
A string representation of the float.

toString

```
public static String toString(double d)
```

Converts the double to a `String`.

Parameters:
`d` - the value to convert.

Returns:
A string representation of the double.

toString

```
public static String toString(java.math.BigDecimal value)
```

Converts the `BigDecimal` to a `String`.

(continued from last page)

Parameters:
`value` - the value to convert.

Returns:
A string representation of the `BigDecimal`.

Throws:
`NullPointerException` - if `value` is a `null` reference.

toPattern

```
public static String toPattern(Calendar calendar,
        boolean includeTimeZone)
```

Creates a pattern describing the date and time format from a given `Calendar` value.

This method checks what `Calendar` date and time fields are set and constructs a pattern from the corresponding pattern elements.

Parameters:
`calendar` - the `Calendar` value to convert.
`includeTimeZone` - `true` to include the time zone designator pattern element `"Z"`; otherwise `false`.

Returns:
The pattern describing the date and time format.

Throws:
`NullPointerException` - if `calendar` is a `null` reference.
`IllegalArgumentException` - if `calendar` has no date or time fields set.

toString

```
public static String toString(Calendar calendar)
```

Converts the `Calendar` to a `String`.

This method calls `toString(Calendar, String)` method with `"yyyy-MM-ddTHH:mm:ssZ"` pattern.

Parameters:
`calendar` - the value to convert.

Returns:
A string representation of the `Calendar` in the date and time format described by `"yyyy-MM-ddTHH:mm:ss.SSSZ"` pattern.

Throws:
`NullPointerException` - if `calendar` is a `null` reference.

toString

```
public static String toString(Calendar calendar,
        String pattern)
```

(continued from last page)

Converts the `Calendar` to a `String` using a specified pattern describing the date and time format.

This method supports ISO 8601, the International Standard for the representation of dates and times.

Supported patterns have the following format:

`[DATE]'T'[TIME][TZD]`,

where `[DATE]` is one of the supported date patterns, `[TIME]` is one of the supported time patterns, `[TZD]` is optional time zone designator. When both date and time parts are present, 'T' indicates the beginning of the time part. For example,

`"yyyy-MM-ddTHH:mm:ss.SSSZ"`.

Supported date and time patterns:

Date patterns	**Time patterns**

`"yyyy"`
`"yyyy-MM"`
`"yyyy-MM-dd"`
	`"HH"`
	`"HH:mm"`
	`"HH:mm:ss"`
	`"HH:mm:ss.S"`
	`"HH:mm:ss.SS"`
	`"HH:mm:ss.SSS"`

Elements used in supported patterns:

Element	**Description**
yyyy	four-digit year
MM	two-digit month
dd	two-digit day of month
hh	two digits of hour (00 through 23)
mm	two digits of minute (00 through 59)
ss	two digits of second (00 through 59)
S, SS or SSS	one, two or three digits representing a decimal fraction of a second
Z	`"Z"` or `"+HH:mm"` or `"-HH:mm"` representing time zone designator

Parameters:

`calendar` - the value to convert.

`pattern` - the pattern describing the date and time format.

Returns:

A string representation of the `Calendar` in the date and time format specified by the `pattern`.

Throws:

`NullPointerException` - if `calendar` or `format` is a `null` reference.

`IllegalArgumentException` - if the specified pattern is not a valid date and time format or includes a pattern element for which the corresponding calendar field is not set or includes the year pattern element `yyyy` and calendar year field has more than four year digits.

toBoolean

```
public static boolean toBoolean(String s)
  throws java.text.ParseException
```

Converts the `String` to a boolean equivalent.

(continued from last page)

Parameters:
`s` - the string to convert.

Returns:
A boolean value, that is, `true` or `false`.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`ParseException` - if `s` s does not represent a `boolean` value.

toChar

```
public static char toChar(String s)
  throws java.text.ParseException
```

Converts the `String` to a char equivalent.

The method supports conversion of strings that represent a character encoded as string in the "_xHHHH_" format, where "HHHH" is the four-digit hexadecimal UCS-2 representation of the character.

Parameters:
`s` - the string to convert.

Returns:
A char equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`ParseException` - if `s` is an empty string or does not represent a char value.

toByte

```
public static byte toByte(String s)
```

Converts the `String` representing signed byte value to a byte equivalent.

Parameters:
`s` - the string to convert.

Returns:
A byte equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable byte.

toUByte

```
public static byte toUByte(String s)
```

Converts the `String` representing unsigned byte value to a byte equivalent.

Parameters:
`s` - the string to convert.

Returns:
A byte equivalent of the string.

Throws:

(continued from last page)

`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable byte.

---

toShort

`public static short **toShort**(String s)`

Converts the `String` representing signed short value to a short equivalent.

Parameters:
`s` - the string to convert.

Returns:
A short equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable short.

---

toUShort

`public static short **toUShort**(String s)`

Converts the `String` representing unsigned short value to a short equivalent.

Parameters:
`s` - the string to convert.

Returns:
A short equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable short.

---

toInt

`public static int **toInt**(String s)`

Converts the `String` representing signed int value to a int equivalent.

Parameters:
`s` - the string to convert.

Returns:
An int equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable int.

---

toUInt

`public static int **toUInt**(String s)`

Converts the `String` representing unsigned int value to a int equivalent.

Parameters:
`s` - the string to convert.

(continued from last page)

Returns:
An int equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable int.

toLong

`public static long **toLong**(String s)`

Converts the `String` representing signed long value to a long equivalent.

Parameters:
`s` - the string to convert.

Returns:
A long equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable long.

toULong

`public static long **toULong**(String s)`

Converts the `String` representing unsigned long value to a long equivalent.

Parameters:
`s` - the string to convert.

Returns:
A long equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable long.

toBigInteger

`public static java.math.BigInteger **toBigInteger**(String s)`

Converts the `String` to a `BigInteger` equivalent.

Parameters:
`s` - the string to convert.

Returns:
A `BigInteger` equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if `s` is not a valid representation of a `BigInteger`.

(continued from last page)

toFloat

`public static float **toFloat**(String s)`

Converts the `String` to a float equivalent.

Parameters:
`s` - the string to convert.

Returns:
A float equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable float.

---

toDouble

`public static double **toDouble**(String s)`

Converts the `String` to a double equivalent.

Parameters:
`s` - the string to convert.

Returns:
A double equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if the string does not contain a parsable double.

---

toBigDecimal

`public static java.math.BigDecimal **toBigDecimal**(String s)`

Converts the `String` to a `BigDecimal` equivalent.

Parameters:
`s` - the string to convert.

Returns:
A `BigDecimal` equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`NumberFormatException` - if `s` is not a valid representation of a `BigDecimal`.

---

toCalendar

`public static Calendar **toCalendar**(String s)`

(continued from last page)

Converts the `String` to a `Calendar` equivalent.

This method supports ISO 8601, the International Standard for the representation of dates and times.

Valid date and time string formats are based on supported patterns and are composed of date and/or time parts followed by an optional time zone designator format part. When both date and time parts are present, 'T' character indicates the beginning of the time part.

See `toString(Calendar, String)` method for the supported patterns.

Parameters:
`s` - the string to convert.

Returns:
A `Calendar` equivalent of the string.

Throws:
`NullPointerException` - if `s` is a `null` reference.
`IllegalArgumentException` - if the specified string is not parsable date and time string.

Package com.imis.util.logging

com.imis.util.logging
Class DetailedFormatter

```
java.lang.Object
  |
  +-java.util.logging.Formatter
      |
      +-com.imis.util.logging.DetailedFormatter
```

public class **DetailedFormatter**
extends Formatter

Prints a detailed summary of the LogRecord that includes log entry date, time, thread, class and method name, followed by the message.
Author:
Robert Petek

Constructor Summary

`public`	`DetailedFormatter(Handler handler)` Initializes a new instance of the `DetailedFormatter` class.
`public`	`DetailedFormatter(Handler handler, char delimiter)` Initializes a new instance of the `DetailedFormatter` class.

Method Summary

`String`	`format(LogRecord record)` Formats the given `LogRecord`.

Methods inherited from class `java.util.logging.Formatter`

`format, formatMessage, getHead, getTail`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

DetailedFormatter

public **DetailedFormatter**(Handler handler)

Initializes a new instance of the `DetailedFormatter` class.

Parameters:
`handler` - a `java.util.logging.Handler` object.

(continued from last page)

DetailedFormatter

```
public DetailedFormatter(Handler handler,
                         char delimiter)
```

Initializes a new instance of the `DetailedFormatter` class.

Parameters:
`handler` - a `java.util.logging.Handler` object.
`delimiter` - a delimiter character.

Methods

format

```
public String format(LogRecord record)
```

Formats the given `LogRecord`.

Parameters:
`record` - the log record to be formatted.

Returns:
The formatted log record.

com.imis.util.logging
Class Logger

```
java.lang.Object
  |
  +-java.util.logging.Logger
      |
      +-com.imis.util.logging.Logger
```

public class **Logger**
extends Logger

Defines a base class for IMiS loggers.

Fields inherited from class `java.util.logging.Logger`
`global`

Constructor Summary

`public`	`Logger(String name, String resourceBundleName)` Initializes a new instance of the `Logger` class.

Method Summary

`void`	`addHandler(Handler handler)` Add a log `Handler` to receive logging messages and enables the logger.
`void`	`debug(Object o, String sourceMethod, String msg)` Log a debug message with no arguments.
`void`	`debug(Object o, String sourceMethod, String msg, Object[] params)` Log a debug message with an array of object arguments.
`void`	`entering(Object o, String sourceMethod)` Log a method entry.
`void`	`entering(Object o, String sourceMethod, Object param1)` Log a method entry, with one parameter.
`void`	`entering(Object o, String sourceMethod, Object[] params)` Log a method entry, with an array of parameters.
`void`	`entering(String sourceClass, String sourceMethod)` Log a method entry.
`void`	`entering(String sourceClass, String sourceMethod, Object param1)` Log a method entry, with one parameter.
`void`	`entering(String sourceClass, String sourceMethod, Object[] params)` Log a method entry, with an array of parameters.
`void`	`error(Object o, String sourceMethod, String msg)` Log an error message with no arguments.

`void`	`error(Object o, String sourceMethod, String msg, Object[] params)` Log an error message with an array of object arguments.
`void`	`error(Object o, String sourceMethod, Throwable e)` Log an error message with associated `Throwable` information.
`void`	`exiting(Object o, String sourceMethod)` Log a method return.
`void`	`exiting(Object o, String sourceMethod, Object result)` Log a method return, with result object.
`void`	`exiting(String sourceClass, String sourceMethod)` Log a method return.
`void`	`exiting(String sourceClass, String sourceMethod, Object result)` Log a method return, with result object.
`void`	`info(Object o, String sourceMethod, String msg)` Log an info message with no arguments.
`void`	`info(Object o, String sourceMethod, String msg, Object[] params)` Log an info message with an array of object arguments.
`void`	`log(LogRecord record)` Logs a `LogRecord`.
`void`	`logEntry(Level level, String sourceClass, String sourceMethod, String msg)` Log a message, specifying source class and method, with no arguments.
`void`	`logEntry(Level level, String sourceClass, String sourceMethod, String msg, Object param1)` Log a message, specifying source class and method, with a single object parameter to the log message.
`void`	`logEntry(Level level, String sourceClass, String sourceMethod, String msg, Object[] params)` Log a message, specifying source class and method, with an array of object arguments.
`void`	`logError(Level level, String sourceClass, String sourceMethod, Throwable thrown)` Log a message, specifying source class and method, with associated `Throwable` information.
`void`	`removeHandler(Handler handler)` Remove a log `Handler` and disables the logger, if no `Handlers` associated with this logger.
`void`	`warn(Object o, String sourceMethod, String msg)` Log a warning message with no arguments.
`void`	`warn(Object o, String sourceMethod, String msg, Object[] params)` Log a warning message with an array of object arguments.
`void`	`warn(Object o, String sourceMethod, Throwable e)` Log a warning message with associated `Throwable` information.

Methods inherited from class `java.util.logging.Logger` addHandler, config, entering, entering, entering, exiting, exiting, fine, finer, finest, getAnonymousLogger, getAnonymousLogger, getFilter, getHandlers, getLevel, getLogger, getLogger, getName, getParent, getResourceBundle, getResourceBundleName, getUseParentHandlers, info, isLoggable, log, log, log, log, log, logp, logp, logp, logp, logrb, logrb, logrb, logrb, removeHandler, setFilter, setLevel, setParent, setUseParentHandlers, severe, throwing, warning

Methods inherited from class `java.lang.Object`

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

Logger

```
public Logger(String name,
              String resourceBundleName)
```

Initializes a new instance of the `Logger` class.

This constructor sets the default log level to `INFO`.

Parameters:

`name` - a name for the logger. This should be a dot-separated name and should normally be based on the package name or class name of the subsystem. It may be `null` for anonymous Loggers.
`resourceBundleName` - the name of `ResourceBundle` to be used for localizing messages for this logger. May be `null` if none of the messages require localization.

Methods

addHandler

```
public void addHandler(Handler handler)
```

Add a log `Handler` to receive logging messages and enables the logger.

By default, `Loggers` also send their output to their parent logger. Typically the root `Logger` is configured with a set of `Handlers` that essentially act as default handlers for all loggers.

Parameters:

`handler` - a `java.util.logging.Handler` instance.

Throws:

`SecurityException` - if a security manager exists and if the caller does not have `LoggingPermission("control")`.

removeHandler

```
public void removeHandler(Handler handler)
```

Remove a log `Handler` and disables the logger, if no `Handlers` associated with this logger.

Returns silently if the given `Handler` is not found.

Parameters:

`handler` - a `java.util.logging.Handler` instance.

(continued from last page)

Throws:

`SecurityException` - if a security manager exists and if the caller does not have `LoggingPermission("control")`.

log

```
public void log(LogRecord record)
```

Logs a `LogRecord`.

All the other logging methods in this class call through this method to actually perform any logging. Subclasses can override this single method to capture all log activity.

Parameters:

`record` - the `LogRecord` to be published.

logEntry

```
public void logEntry(Level level,
         String sourceClass,
         String sourceMethod,
         String msg)
```

Log a message, specifying source class and method, with no arguments.

If the logger is currently enabled for the given message level then the given message is forwarded to all the registered output `Handler` objects.

Parameters:

`level` - one of the message level identifiers, e.g. `SEVERE`
`sourceClass` - name of class that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)

logEntry

```
public void logEntry(Level level,
         String sourceClass,
         String sourceMethod,
         String msg,
         Object param1)
```

Log a message, specifying source class and method, with a single object parameter to the log message.

If the logger is currently enabled for the given message level then a corresponding `LogRecord` is created and forwarded to all the registered output `Handler` objects.

Parameters:

`level` - one of the message level identifiers, e.g. `SEVERE`
`sourceClass` - name of class that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)
`param1` - parameter to the log message.

logEntry

```
public void logEntry(Level level,
         String sourceClass,
         String sourceMethod,
         String msg,
         Object[] params)
```

(continued from last page)

Log a message, specifying source class and method, with an array of object arguments.

If the logger is currently enabled for the given message level then a corresponding `LogRecord` is created and forwarded to all the registered output `Handler` objects.

Parameters:

`level` - one of the message level identifiers, e.g. `SEVERE`
`sourceClass` - name of class that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)
`params` - array of parameters to the message

logError

```
public void logError(Level level,
         String sourceClass,
         String sourceMethod,
         Throwable thrown)
```

Log a message, specifying source class and method, with associated `Throwable` information.

If the logger is currently enabled for the given message level then a corresponding `LogRecord` is created and forwarded to all the registered output `Handler` objects.

Note that the thrown argument is stored in the `LogRecord` thrown property, rather than the `LogRecord` parameters property. Thus is it processed specially by output `Formatters` and is not treated as a formatting parameter to the `LogRecord` message property.

Parameters:

`level` - one of the message level identifiers, e.g. `SEVERE`
`sourceClass` - name of class that issued the logging request
`sourceMethod` - name of method that issued the logging request
`thrown` - `Throwable` associated with log message.

entering

```
public void entering(String sourceClass,
         String sourceMethod)
```

Log a method entry.

This is a convenience method that can be used to log entry to a method. A `LogRecord` with message **"ENTRY"**, log level `FINER`, and the given `sourceMethod` and `sourceClass` is logged.

Parameters:

`sourceClass` - name of class that issued the logging request
`sourceMethod` - name of method that is being entered

entering

```
public void entering(Object o,
         String sourceMethod)
```

Log a method entry.

This is a convenience method that can be used to log entry to a method. A `LogRecord` with message **"ENTRY"**, log level `FINER`, and the given `sourceMethod` and `sourceClass` is logged.

Parameters:

`o` - source object that issued the logging request
`sourceMethod` - name of method that is being entered

entering

```
public void entering(String sourceClass,
         String sourceMethod,
         Object param1)
```

Log a method entry, with one parameter.

This is a convenience method that can be used to log entry to a method. A `LogRecord` with message **"ENTRY {0}"**, log level `FINER`, and the given `sourceMethod` `sourceClass`, and parameter is logged.

Parameters:

`sourceClass` - name of class that issued the logging request

`sourceMethod` - name of method that is being entered

`param1` - parameter to the method being entered

entering

```
public void entering(Object o,
         String sourceMethod,
         Object param1)
```

Log a method entry, with one parameter.

This is a convenience method that can be used to log entry to a method. A `LogRecord` with message **"ENTRY {0}"**, log level `FINER`, and the given `sourceMethod` `sourceClass`, and parameter is logged.

Parameters:

`o` - source object that issued the logging request

`sourceMethod` - name of method that is being entered

`param1` - parameter to the method being entered

entering

```
public void entering(String sourceClass,
         String sourceMethod,
         Object[] params)
```

Log a method entry, with an array of parameters.

This is a convenience method that can be used to log entry to a method. A `LogRecord` with message **"ENTRY {0} {1} .. {n}"**, log level `FINER`, and the given `sourceMethod` `sourceClass`, and **n** parameters is logged.

Parameters:

`sourceClass` - name of class that issued the logging request

`sourceMethod` - name of method that is being entered

`params` - parameters to the method being entered

entering

```
public void entering(Object o,
         String sourceMethod,
         Object[] params)
```

Log a method entry, with an array of parameters.

This is a convenience method that can be used to log entry to a method. A `LogRecord` with message **"ENTRY {0} {1} .. {n}"**, log level `FINER`, and the given `sourceMethod` `sourceClass`, and **n** parameters is logged.

Parameters:

`o` - source object that issued the logging request

(continued from last page)

`sourceMethod` - name of method that is being entered

`params` - parameters to the method being entered

exiting

```
public void exiting(String sourceClass,
         String sourceMethod)
```

Log a method return.

This is a convenience method that can be used to log returning from a method. A `LogRecord` with message `"RETURN"`, log level `FINER`, and the given `sourceMethod` and `sourceClass` is logged.

Parameters:

`sourceClass` - name of class that issued the logging request

`sourceMethod` - name of method that is being exited

exiting

```
public void exiting(Object o,
         String sourceMethod)
```

Log a method return.

This is a convenience method that can be used to log returning from a method. A `LogRecord` with message `"RETURN"`, log level `FINER`, and the given `sourceMethod` and `sourceClass` is logged.

Parameters:

`o` - source object that issued the logging request

`sourceMethod` - name of method that is being exited

exiting

```
public void exiting(String sourceClass,
         String sourceMethod,
         Object result)
```

Log a method return, with result object.

This is a convenience method that can be used to log returning from a method. A `LogRecord` with message `"RETURN"`, log level `FINER`, and the given `sourceMethod`, `sourceClass`, and result object is logged.

Parameters:

`sourceClass` - name of class that issued the logging request

`sourceMethod` - name of method that is being exited

`result` - `Object` that is being returned

exiting

```
public void exiting(Object o,
         String sourceMethod,
         Object result)
```

Log a method return, with result object.

This is a convenience method that can be used to log returning from a method. A `LogRecord` with message `"RETURN"`, log level `FINER`, and the given `sourceMethod`, `sourceClass`, and result object is logged.

Parameters:

`o` - source object that issued the logging request

`sourceMethod` - name of method that is being exited

`result` - `Object` that is being returned

debug

```
public void debug(Object o,
         String sourceMethod,
         String msg)
```

Log a debug message with no arguments.

Parameters:
`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)

debug

```
public void debug(Object o,
         String sourceMethod,
         String msg,
         Object[] params)
```

Log a debug message with an array of object arguments.

Parameters:
`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)
`params` - array of parameters to the message

info

```
public void info(Object o,
         String sourceMethod,
         String msg)
```

Log an info message with no arguments.

Parameters:
`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)

info

```
public void info(Object o,
         String sourceMethod,
         String msg,
         Object[] params)
```

Log an info message with an array of object arguments.

Parameters:
`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)
`params` - array of parameters to the message

(continued from last page)

warn

```
public void warn(Object o,
         String sourceMethod,
         String msg)
```

Log a warning message with no arguments.

Parameters:
`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)

warn

```
public void warn(Object o,
         String sourceMethod,
         String msg,
         Object[] params)
```

Log a warning message with an array of object arguments.

Parameters:
`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)
`params` - array of parameters to the message

warn

```
public void warn(Object o,
         String sourceMethod,
         Throwable e)
```

Log a warning message with associated `Throwable` information.

Parameters:
`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`e` - `Throwable` associated with log message.

error

```
public void error(Object o,
         String sourceMethod,
         String msg)
```

Log an error message with no arguments.

Parameters:
`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)

error

```
public void error(Object o,
         String sourceMethod,
         String msg,
         Object[] params)
```

(continued from last page)

Log an error message with an array of object arguments.

Parameters:

`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`msg` - the string message (or a key in the message catalog)
`params` - array of parameters to the message

error

```
public void error(Object o,
         String sourceMethod,
         Throwable e)
```

Log an error message with associated `Throwable` information.

Parameters:

`o` - source object that issued the logging request
`sourceMethod` - name of method that issued the logging request
`e` - `Throwable` associated with log message.

Index