Paginated Javadoc Test

Packages

Package com.privity.common.exceptions

ConfigurationException

extends com.privity.common.exceptions.SystemException

Public Constructors

ConfigurationException()

public **ConfigurationException**(
String s .
)

ConfigurationException()

public **ConfigurationException**(
String s, .
Throwable t .
)

Author(s):
andy pruitt *rapruitt@earthlink.net*

SystemException

extends java.lang.RuntimeException

Direct Known Subclasses:
com.privity.common.exceptions.ConfigurationException

Public Constructors

SystemException()
public **SystemException**(
String `s` .
)

SystemException()
public **SystemException**(
String `s,` .
Throwable `t` .
)

Public Methods

getRootException()
public Throwable **getRootException**( )
Returns:
`Throwable`

getThrowable()
public Throwable **getThrowable**( )
Returns:
`Throwable`

Author(s):
andy pruitt *rapruitt@earthlink.net*

Package com.privity.common.utils

Documentation for the utils package. This package is for basic utility classes.

CreationDateComparator

implements java.util.Comparator

Public Constructors

CreationDateComparator()
public **CreationDateComparator**( )

Public Methods

compare()
public int **compare**(
Object `o1,`
Object `o2`
)
Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

The implementor must ensure that `sgn(compare(x, y)) == -sgn(compare(y, x))` for all `x` and `y`. (This implies that `compare(x, y)` must throw an exception if and only if `compare(y, x)` throws an exception.)

The implementor must also ensure that the relation is transitive: `((compare(x, y)>0) && (compare(y, z)>0))` implies `compare(x, z)>0`.

Finally, the implementer must ensure that `compare(x, y)==0` implies that `sgn(compare(x, z))==sgn(compare(y, z))` for all `z`.

It is generally the case, but *not* strictly required that `(compare(x, y)==0) == (x.equals(y))`. Generally speaking, any comparator that violates this condition should clearly indicate this fact. The recommended language is "Note: this comparator imposes orderings that are inconsistent with equals."
Returns:
`int`

Throws:

`ClassCastException`	if the arguments' types prevent them from being compared by this Comparator.

.

Author(s):
andy pruitt *rapruitt@earthlink.net*

Money

implements java.lang.Comparable

Represents an amount of money. The default currency is USD

Public Fields

String USD	*American Dollars.*
	.

Public Constructors

Money()

public **Money**(
long `amountInCents` .
)

Money()

public **Money**(
long `amountInCents,` .
short `roundingPolicy` .
)

Money()

public **Money**(
long `amountInCents,` .
String `theCurrency` .
)

Public Methods

add()

public Money **add**(
Money `other` .
)
Returns:

`Money`

compareTo()

public int **compareTo**(
Object o .
)
Returns:
`int`

Throws:

`ClassCastException`	if the specified object's type prevents it from being compared to this Object. Also thrown if the other money object is of a different currency. .

divideBy()

public Money **divideBy**(
long `denominator` .
)
Decide on rounding policy here
Returns:
`Money`

Throws:

`IllegalArgumentException`	if 0 is passed. .

equals()

public boolean **equals**(
Object `obj` .
)
Indicates whether some other object is "equal to" this one.

The `equals` method implements an equivalence relation: <ul> It is *reflexive*: for any reference value `x`, `x.equals(x)` should return `true`. It is *symmetric*: for any reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`. It is *transitive*: for any reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`. It is *consistent*: for any reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons

on the object is modified. For any non-null reference value `x`, `x.equals(null)` should return `false`.

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (x==y has the value `true`).

See Also:
Boolean.hashCode() , Hashtable
Returns:
`boolean`

getAtomicAmount()

public long **getAtomicAmount**( )
Returns:
`long`

getCurrency()

public String **getCurrency**( )
Returns:
`String`

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`.

The general contract of `hashCode` is: <ul> Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application. If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result. It is **not** required that if two objects are unequal according to the [1] method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

1 . See the Object.equals() method.

As much as is reasonably practical, the hashCode method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the JavaTM programming language.)

See Also:
Object.equals() , Hashtable
Returns:
`int`

isSameCurrency()

public boolean **isSameCurrency**(
Money `other` .
)
Returns:
`boolean`

multiply()

public Money **multiply**(
long `multiplier` .
)
Returns:
`Money`

subtract()

public Money **subtract**(
Money `lessThisMuch` .
)
Returns:
`Money`

toString()

public String **toString**( )
Returns:
`String`

Author(s):

andy pruitt *rapruitt@earthlink.net*

NameComparator

implements java.util.Comparator

This Comparator allows you to sort any objects that have a name.

This class is especially useful for sorting entity beans.

See Also:
Named, Comparator, SystemException.getRootException() (page 5) , CreationDateComparator (page 8)

Public Constructors

NameComparator()
public **NameComparator**( )

Public Methods

compare()
public int **compare**(
Object `o1,` .
Object `o2` .
)
Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

The implementor must ensure that `sgn(compare(x, y)) == -sgn(compare(y, x))` for all `x` and `y`. (This implies that `compare(x, y)` must throw an exception if and only if `compare(y, x)` throws an exception.)

a random link [2] another link [3] third link [4] The implementor must also ensure that the relation is transitive: `((compare(x, y)>0) && (compare(y, z)>0))` implies `compare(x, z)>0`.

2 . See the SystemException class on page 5.
3 . See the ConfigurationException.getMessage() method on page 4.
4 . See the compare() method.

Finally, the implementer must ensure that `compare(x, y)==0` implies that `sgn(compare(x, z))==sgn(compare(y, z))` for all `z`.

It is generally the case, but *not* strictly required that `(compare(x, y)==0) == (x.equals(y))`. Generally speaking, any comparator that violates this condition should clearly indicate this fact. The recommended language is "Note: this comparator imposes orderings that are inconsistent with equals."

Returns:

`int`

Throws:

`SystemException`	if a communication error occurs .
`ClassCastException`	if the arguments' types prevent them from being compared by this Comparator. .

Author(s):
andy href pruitt *rapruitt@earthlink.net*

Sequence

Provides the next available value for database ids.

Public Constructors

Sequence()
public **Sequence**( )

Public Methods

getSequence()
public static int **getSequence**(
String `tableName` .
)
Get a new key value. Guarranteed unique.
Returns:
`int`

Throws:
`SystemException` if an error occurs retrieving the next value .

Author(s):
andy pruitt *rapruitt@earthlink.net*