The JDBC(tm) API Version 1.01

August 7, 1996

Part 2 - Classes and Exceptions

This document contains a paper copy of the JDBC API online documentation that is distributed with the JDBC package and is also available on http://splash.javasoft.com/jdbc.

It takes the place of the source code comments that were originally included as part of JDBC specification.

---

package java.sql

Interface Index

- CallableStatement
- Connection
- DatabaseMetaData
- Driver
- PreparedStatement
- ResultSet
- ResultSetMetaData
- Statement

Class Index

- Date
- DriverManager
- DriverPropertyInfo
- Numeric
- Time
- Timestamp
- Types

Exception Index

- DataTruncation
- SQLException
- SQLWarning

Class java.sql.Date

```
java.lang.Object
   |
   +----java.util.Date
           |
           +----java.sql.Date
```

public class **Date**
extends Date

This class is used to represent a subset of the standard java.util.date information. We only deal with days and ignore hours, minutes, and seconds. This lets us represent SQL DATE information.

Constructor Index

- **Date**(int, int, int)
  Construct a Date

Method Index

- **toString**()
  Format a date as yyyy-mm-dd
- **valueOf**(String)
  Convert a formatted string to a Date value

Constructors

- **Date**

```
public Date(int year,
            int month,
            int day)
```

  Construct a Date
  Parameters:
    year - year-1900
    month - 0 to 11
    day - 1 to 31

Methods

- **valueOf**

```
public static Date valueOf(String s)
```

  Convert a formatted string to a Date value
  Parameters:
    s - date in format "yyyy-mm-dd"

Returns:
corresponding Date

- **toString**

```
public String toString()
```

Format a date as yyyy-mm-dd
Returns:
a formatted date String
Overrides:
toString in class Date

Class java.sql.DriverManager

```
java.lang.Object
   |
   +----java.sql.DriverManager
```

---

public class **DriverManager**
extends Object

The DriverManager class provides access to global SQL state.

As part of its initialization the DriverManager class will use the system "jdbc.drivers" property. This should contain a list of classnames for driver classes. The DriverManager class will attempt to load each of these driver classes. For example in your ~/.hotjava/properties file you might specify:
`jdbc.drivers=foo.bah.Driver:wombat.sql.Driver:bad.taste.ourDriver`

Sunsequently when you attempt to open a database connection the DriverManager class will attempt to locate a suitable driver class from amongst these driver classes and from any other driver classes which have been loaded from the same classloader as the current applet.

See Also:
Driver, Connection

Constructor Index

- **DriverManager**()

Method Index

- **deregisterDriver**(Driver)
  Drop a Driver from the DriverManager's list.
- **getConnection**(String)
  Attempt to establish a connection to the given database URL.
- **getConnection**(String, Properties)
  Attempt to establish a connection to the given database URL.
- **getConnection**(String, String, String)
  Attempt to establish a connection to the given database URL.
- **getDriver**(String)
  Attempt to locate a driver that understands the given URL.
- **getDrivers**()
  Return an Enumeration of all the currently loaded JDBC drivers which the current caller has access to.
- **getLoginTimeout**()
  Get the maximum time in seconds that all drivers can wait when attempting to login to a database.
- **getLogStream**()
  Get the logging/tracing PrintStream that is used by the DriverManager and all drivers.
- **println**(String)
  Print a message to the current JDBC log stream
- **registerDriver**(Driver)
  A newly loaded driver class should call registerDriver to make itself known to the DriverManager.
- **setLoginTimeout**(int)
  Set the maximum time in seconds that all drivers can wait when attempting to login to a database.
- **setLogStream**(PrintStream)
  Set the logging/tracing PrintStream that is used by the DriverManager and all drivers.

Constructors

- **DriverManager**

```
public DriverManager()
```

Methods

- **getConnection**

```
public static synchronized Connection getConnection(String url,
                                                    Properties info) throws SQLExc
```

  Attempt to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

  Parameters:

  url - a database url of the form jdbc:*subprotocol*:*subname*

info - a list of arbitrary string tag/value pairs as connection arguments; normally at least a "user" and "password" property should be included

Returns:

a Connection to the URL

- **getConnection**

```
public static synchronized Connection getConnection(String url,
                                                    String user,
                                                    String password) throws SQLExc
```

Attempt to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:

url - a database url of the form jdbc:*subprotocol*:*subname*

user - the database user on whom's behalf the Connection is being made

password - the user's password

Returns:

a Connection to the URL

- **getConnection**

```
public static synchronized Connection getConnection(String url) throws SQLExceptio
```

Attempt to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:

url - a database url of the form jdbc:*subprotocol*:*subname*

Returns:

a Connection to the URL

- **getDriver**

```
public static Driver getDriver(String url) throws SQLException
```

Attempt to locate a driver that understands the given URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:

url - a database url of the form jdbc:*subprotocol*:*subname*

Returns:

a Driver that can connect to the URL

- **registerDriver**

```
public static synchronized void registerDriver(Driver driver) throws SQLException
```

A newly loaded driver class should call registerDriver to make itself known to the DriverManager.

Parameters:

driver - the new JDBC Driver

- **deregisterDriver**

```
public static void deregisterDriver(Driver driver) throws SQLException
```

  Drop a Driver from the DriverManager's list. Applets can only deregister Drivers from their own classloader.
  Parameters:
    driver - the JDBC Driver to drop

- **getDrivers**

```
public static Enumeration getDrivers()
```

  Return an Enumeration of all the currently loaded JDBC drivers which the current caller has access to.

  Note: The classname of a driver can be found using `d.getClass().getName()`
  Returns:
    the list of JDBC Driver's loaded by the caller's class loader

- **setLoginTimeout**

```
public static void setLoginTimeout(int seconds)
```

  Set the maximum time in seconds that all drivers can wait when attempting to login to a database.
  Parameters:
    seconds - the driver login time limit

- **getLoginTimeout**

```
public static int getLoginTimeout()
```

  Get the maximum time in seconds that all drivers can wait when attempting to login to a database.
  Returns:
    the driver login time limit

- **setLogStream**

```
public static void setLogStream(PrintStream out)
```

  Set the logging/tracing PrintStream that is used by the DriverManager and all drivers.
  Parameters:
    out - the new logging/tracing PrintStream; to disable, set to null

- **getLogStream**

```
public static PrintStream getLogStream()
```

  Get the logging/tracing PrintStream that is used by the DriverManager and all drivers.
  Returns:

the logging/tracing PrintStream; if disabled, is null

- **println**

```
public static void println(String message)
```

Print a message to the current JDBC log stream
Parameters:
message - a log or tracing message

Class java.sql.DriverPropertyInfo

```
java.lang.Object
   |
   +----java.sql.DriverPropertyInfo
```

---

public class **DriverPropertyInfo**
extends Object

The DriverPropertyInfo class is only of interest to advanced programmers who need to interact with a Driver via getDriverProperties to discover and supply properties for connections.

Variable Index

- **choices**
  If the value may be selected from a particular set of values, then this is an array of the possible values.
- **description**
  A brief description of the property.
- **name**
  The name of the property.
- **required**
  "required" is true if a value must be supplied for this property during Driver.connect.
- **value**
  "value" specifies the current value of the property, based on a combination of the information supplied to getPropertyInfo, the Java environment, and driver supplied default values.

Constructor Index

- **DriverPropertyInfo**(String, String)

Initial constructor.

Variables

- **name**

```
public String name
```

The name of the property.

- **description**

```
public String description
```

A brief description of the property. This may be null.

- **required**

```
public boolean required
```

"required" is true if a value must be supplied for this property during Driver.connect. Otherwise the property is optional.

- **value**

```
public String value
```

"value" specifies the current value of the property, based on a combination of the information supplied to getPropertyInfo, the Java environment, and driver supplied default values. This may be null if no value is known.

- **choices**

```
public String choices[]
```

If the value may be selected from a particular set of values, then this is an array of the possible values. Otherwise it should be null.

Constructors

- **DriverPropertyInfo**

```
public DriverPropertyInfo(String name,
                          String value)
```

Initial constructor. The name is the name of the property and the value is the current value, which may be null.

Class java.sql.Numeric

```
java.lang.Object
   |
   +----java.lang.Number
           |
           +----java.sql.Numeric
```

---

public final class **Numeric**
extends Number

The Numeric class represents arbitrary precision and scale numbers. It performs fixed point arithmetic.

The Numeric class should be used for values that require high precision fixed point or integer computation. Examples of this are money values, security key values and values stored in databases using the high precision SQL NUMERIC or DECIMAL type.

The scale of Numeric objects may be declared arbitrarily large. Calculations on Numeric objects always return a new Numeric object with the same scale as the Numeric that performed the calculation. Most operations are carried out to one extra decimal place and rounded to the desired scale. Rounding behavior for the class as a whole is controlled by changing the classes Rounding Value.

Numeric objects are composed of three properties:

- Scale: The number of digits to the right of the decimal point.
- Sign: Positive or negative.
- Value: A vector of integer digits of radix 2^30.

Variable Index

- **ONE**
- **ZERO**

Constructor Index

- **Numeric**(double, int)
  Construct a Numeric from a double given a double value and an integer scale.
- **Numeric**(int)
  Construct a Numeric from an integer given an integer value.
- **Numeric**(int, int)

Construct a Numeric from an integer given an integer value and an integer scale.

- **Numeric**(long)
  Construct a Numeric from an integer given an integer value.
- **Numeric**(long, int)
  Construct a Numeric from a long integer given a long value and an integer scale.
- **Numeric**(Numeric)
  Construct a Numeric from another Numeric.
- **Numeric**(Numeric, int)
  Given an existing Numeric and an integer scale value, construct a new Numeric with the scale adjusted accordingly.
- **Numeric**(String)
  Creates a numeric from a string.
- **Numeric**(String, int)
  Construct a Numeric given a String and an integer scale.

Method Index

- **add**(Numeric)
  Returns the arithmetic sum of the Numeric and the argument.
- **createFromByteArray**(byte[])
  Return a Numeric created from the given byte array.
- **createFromIntegerArray**(int[])
  Return a Numeric created from the given byte array.
- **createFromRadixString**(String, int)
  Return a Numeric created from the given string and radix.
- **createFromScaled**(long, int)
  Return a numeric value created by dividing the argument by 10**scale.
- **divide**(Numeric)
  Returns the arithmentic quotient calculated by dividing the Numeric by the argument.
- **doubleValue**()
  Convert the Numeric value to the Java built-in type of double.
- **dump**(String)
- **equals**(Object)
  Returns true if the arithmetic value of the Numeric equals the argument.
- **floatValue**()
  Convert the Numeric value to the Java built-in type of float.
- **getRoundingValue**()
  Returns the value of the roundingValue which is used in rounding operations.
- **getScale**()
  Return the number of digits to the right of the decimal point.
- **getScaled**()
  Return the value multiplied by 10**scale.
- **greaterThan**(Numeric)
  Returns true if the arithmetic value of the Numeric is greater than the argument.
- **greaterThanOrEquals**(Numeric)
  Returns true if the arithmetic value of the Numeric is greater than or equals the argument.

- **hashCode**()
  Returns an integer hashcode for the Numeric.
- **integerDivide**(Numeric)
  Returns an array of two Numeric objects.
- **intValue**()
  Convert the Numeric value to the Java built-in type of int.
- **isProbablePrime**()
  This method tests for primality, first by attempting to divide by a few small primes, then by using the Rabin probabilistic primality test 50 time.
- **lessThan**(Numeric)
  Returns true if the arithmetic value of the Numeric is less than the argument.
- **lessThanOrEquals**(Numeric)
  Returns true if the arithmetic value of the Numeric is less than or equals the argument.
- **longValue**()
  Convert the Numeric value to the Java built-in type of long.
- **modExp**(Numeric, Numeric)
  Modular exponentiation
- **modInverse**(Numeric)
  Multiplicative inverse k^-1 mod q
- **multiply**(Numeric)
  Returns the arithmetic product of the object Numeric and the argument.
- **pi**(int)
  Returns pi to the number of places given by the argument.
- **pow**(int)
  Returns this to the e'th power.
- **random**(int, Random)
- **remainder**(Numeric)
  Returns the arithmentic remainder calculated by dividing this Numeric object by the argument.
- **setRoundingValue**(int)
  Sets the value of the roundingValue which is used in rounding operations.
- **setScale**(int)
  Returns a numeric copied from the current object with the scale adjusted as specified by the argument.
- **shiftLeft**(int)
  Shift left sh bits.
- **shiftRight**(int)
  Shift the Numeric right by sh bits.
- **significantBits**()
  Returns the number of significant bits.
- **sqrt**()
  Returns the square root of this Numeric.
- **subtract**(Numeric)
  Returns the arithmetic difference between the Numeric and the argument.
- **toString**()
  Convert the Numeric value to the Java built-in type of String.
- **toString**(int)
  Convert to a String.

Variables

- **ZERO**

```
public final static Numeric ZERO
```

- **ONE**

```
public final static Numeric ONE
```

Constructors

- **Numeric**

```
public Numeric(String s)
```

  Creates a numeric from a string. The string may contain a sign and a decimal point, e.g "-55.12". Spaces and other non-numeric characters are ignored.
  Parameters:
  s - the string used to initialize the Numeric.

- **Numeric**

```
public Numeric(String s,
               int scale)
```

  Construct a Numeric given a String and an integer scale. The scale represents the desired number of places to the right of the decimal point.

  The String may contain a sign and a decimal point, e.g. "-55.12". The scale must be an integer with value greater than or equal to zero. If the scale differs from the implicit decimal point given in the String the value is adjusted to the given scale. This may involve rounding.

  For example, Numeric("99.995",2) would result in a numeric with a scale of 2 and a value of "100.00". If the String contains no decimal point, then the scale is determined solely by the given integer scale. No implicit decimal point is assumed. For example, the constructor Numeric("123",2) creates a Numeric with a scale of 2 and a value of "123.00".

  Note: Rounding is controlled by the roundingIndex function.
  Parameters:
  s - the string used to initialize the Numeric.
  scale - the value greater than or equal to zero representing the desired number of digits to the right of the decimal point.

- **Numeric**

```
public Numeric(int x,
               int scale)
```

Construct a Numeric from an integer given an integer value and an integer scale. The scale is set to the value specified. No implicit decimal point is assumed, i.e., if the integer value is "1234" and the scale 2, the resulting numeric will be "1234.00.

Parameters:

x - The int used to initialize the new Numeric.
scale - The desired number of digits after the decimal point.

- **Numeric**

```
public Numeric(int x)
```

Construct a Numeric from an integer given an integer value. The scale is set to zero.

Parameters:

x - The int used to initialize the new Numeric.

- **Numeric**

```
public Numeric(long x,
               int scale)
```

Construct a Numeric from a long integer given a long value and an integer scale. The scale is set to the value specified. No implicit decimal point is assumed, i.e., if the long value is "1234" and the scale 2, the resulting numeric will be "1234.00.

Parameters:

x - The long value used to initialize the new Numeric.
scale - The desired number of digits after the decimal point.

- **Numeric**

```
public Numeric(long x)
```

Construct a Numeric from an integer given an integer value. The scale is set to zero.

Parameters:

x - The int used to initialize the new Numeric.

- **Numeric**

```
public Numeric(double x,
               int scale)
```

Construct a Numeric from a double given a double value and an integer scale. The scale is set to the value specified. No implicit decimal point is assumed, i.e., if the double value is "1234" and the scale 2, the resulting numeric will be "1234.00. If the double value is "1234.5" and the scale 2 the value will be "1234.50". Rounding may occur during this conversion.

Parameters:

x - The double value used to initialize the new Numeric.
scale - The desired number of digits after the decimal point.

- **Numeric**

```
public Numeric(Numeric x)
```

Construct a Numeric from another Numeric. The scale and value of the new Numeric are copied from the Numeric given in the argument.

Parameters:

x - The Numeric used to initialize the new Numeric.

- **Numeric**

```
public Numeric(Numeric x,
               int scale)
```

Given an existing Numeric and an integer scale value, construct a new Numeric with the scale adjusted accordingly. The scale and value of the new Numeric are copied from the argument Numeric. The scale is then adjusted to the scale given as a parameter. This may result in rounding of the value.

Parameters:

x - The Numeric to copy.

scale - An integer representing the desired * scale of the new Numeric.

Methods

- **setRoundingValue**

```
public static void setRoundingValue(int val)
```

Sets the value of the roundingValue which is used in rounding operations. The roundingValue is a digit between 0 and 9 that determines rounding behavior. When the scale of a Numeric is reduced either by request or in a calculation, rounding is performed if the value of the scale + 1 digit to the right of the decimal point is greater than the rounding index.

For example, if the number is 1.995, and the scale is set to 2 and the roundingValue is 4, then the number would be rounded to 2.00. The default roundingValue is 4. A roundingValue of 9 cancels rounding.

Parameters:

val - The value between 0 and 9 of the desired roundingValue.

- **getRoundingValue**

```
public static int getRoundingValue()
```

Returns the value of the roundingValue which is used in rounding operations. The roundingValue is a digit between 0 and 9 that determines rounding behavior. When the scale of a Numeric is reduced either by request or in a calculation, rounding is performed if the value of the scale + 1 digit to the right of the decimal point is greater than the rounding index.

For example, if the number is 1.995, and the scale is set to 2 and the RoundingValue is 4, then the number would be rounded to 2.00. The default roundingValue is 4. A roundingValue of 9 cancels

rounding.
Returns:
rounding value

- **intValue**

```
public int intValue()
```

Convert the Numeric value to the Java built-in type of int. This conversion may result in a loss of precision. Digits after the decimal point are dropped.
Returns:
An integer representation of the Numeric value.
Overrides:
intValue in class Number

- **longValue**

```
public long longValue()
```

Convert the Numeric value to the Java built-in type of long. This conversion may result in a loss of precision. Digits after the decimal point are dropped.
Returns:
A long integer representation of the Numeric value.
Overrides:
longValue in class Number

- **floatValue**

```
public float floatValue()
```

Convert the Numeric value to the Java built-in type of float. This conversion may result in a loss of precision.
Returns:
A float representation of the Numeric value.
Overrides:
floatValue in class Number

- **doubleValue**

```
public double doubleValue()
```

Convert the Numeric value to the Java built-in type of double. This conversion may result in a loss of precision.
Returns:
A double representation of the Numeric value.
Overrides:
doubleValue in class Number

- **toString**

```
public String toString()
```

Convert the Numeric value to the Java built-in type of String. Negative numbers are represented by a leading "-". No sign is added for positive numbers.

Returns:

A String representation of the Numeric value.

Overrides:

toString in class Object

- **getScale**

```
public int getScale()
```

Return the number of digits to the right of the decimal point.

Returns:

An integer value representing the number of decimal places to the right of the decimal point.

- **getScaled**

```
public long getScaled()
```

Return the value multiplied by 10**scale. Precision may be lost. Thus, if a currency value was being kept with scale "2" as "5.04", the getScaled function would return the long integer "504".

Returns:

The scaled value as a long.

- **createFromScaled**

```
public static Numeric createFromScaled(long scaled,
                                       int s)
```

Return a numeric value created by dividing the argument by 10**scale. Thus, createFromScaled(504,2) would create the value "5.04".

Parameters:

scaled - The scaled value as a long.

s - The desired scale value

Returns:

A new numeric.

- **add**

```
public Numeric add(Numeric n)
```

Returns the arithmetic sum of the Numeric and the argument. The scale of the sum is determined by this object not by the argument.

Parameters:

n - The Numeric to add.

Returns:

The sum as a Numeric.

- **subtract**

```
public Numeric subtract(Numeric n)
```

Returns the arithmetic difference between the Numeric and the argument. The scale of the sum is determined by this object not by the argument.

Parameters:

n - The Numeric to subtract.

Returns:

The difference as a Numeric.

- **multiply**

```
public Numeric multiply(Numeric x)
```

Returns the arithmetic product of the object Numeric and the argument. The scale of the product is determined by this object not by the argument.

Parameters:

x - The multiplier as a Numeric.

Returns:

The product as a Numeric.

- **divide**

```
public Numeric divide(Numeric x)
```

Returns the arithmentic quotient calculated by dividing the Numeric by the argument. The scale of the quotient is determined by this object not by the argument.

Parameters:

x - The divisor as a Numeric.

Returns:

The quotient as a Numeric.

- **integerDivide**

```
public Numeric[] integerDivide(Numeric x)
```

Returns an array of two Numeric objects. The first element in the array holds the quotient value resulting from the arithmetic division of this Numeric object by the argument. The second element in the array holds the remainder. If the scale of either the divisor or dividend is not zero, then the remainder will be set to zero.

Parameters:

x - The divisor as a Numeric.

Returns:

An array of two Numeric objects containing the quotient and remainder of the division.

- **remainder**

```
public Numeric remainder(Numeric x)
```

Returns the arithmentic remainder calculated by dividing this Numeric object by the argument. If either the dividend or the divisor has a scale other than zero, then the remainder is set to zero.

Parameters:

x - The divisor as a Numeric.

Returns:

The remainder as a Numeric.

- **equals**

```
public boolean equals(Object obj)
```

Returns true if the arithmetic value of the Numeric equals the argument.

Parameters:

obj - The object to compare.

Returns:

The boolean result of the comparison.

Overrides:

equals in class Object

- **lessThan**

```
public boolean lessThan(Numeric x)
```

Returns true if the arithmetic value of the Numeric is less than the argument.

Parameters:

x - The object to compare.

Returns:

The boolean result of the comparison.

- **lessThanOrEquals**

```
public boolean lessThanOrEquals(Numeric x)
```

Returns true if the arithmetic value of the Numeric is less than or equals the argument.

Parameters:

x - The object to compare.

Returns:

The boolean result of the comparison.

- **greaterThan**

```
public boolean greaterThan(Numeric x)
```

Returns true if the arithmetic value of the Numeric is greater than the argument.

Parameters:

x - The object to compare.

Returns:

The boolean result of the comparison.

- **greaterThanOrEquals**

```
public boolean greaterThanOrEquals(Numeric x)
```

  Returns true if the arithmetic value of the Numeric is greater than or equals the argument.
  Parameters:
    x - The object to compare.
  Returns:
    The boolean result of the comparison.

- **hashCode**

```
public int hashCode()
```

  Returns an integer hashcode for the Numeric. Note that the code returned for 03.00 will equal the code for 3.0.
  Returns:
    The hashcode as an integer.
  Overrides:
    hashCode in class Object

- **setScale**

```
public Numeric setScale(int scale)
```

  Returns a numeric copied from the current object with the scale adjusted as specified by the argument. Sets the number of the digits to the right of the decimal point. The value of the Numeric will be adjusted accordingly. Note that changing the scale to a smaller value may result in rounding the value.
  Parameters:
    scale - the character to be converted
  Returns:
    A Numeric with the adjusted scale.
  See Also:
    roundingValue

- **createFromRadixString**

```
public static Numeric createFromRadixString(String s,
                                            int radix)
```

  Return a Numeric created from the given string and radix. Decimal places are ignored and the scale is set to zero. The string may contain a sign.
  Parameters:
    s - A string containing the intended value of the numeric.
    radix - The base of the string representation.

- **createFromByteArray**

```
public static Numeric createFromByteArray(byte b[])
```

Return a Numeric created from the given byte array. The array is considered a string of bits with the most significant bit at the 0th position..

Parameters:

b - An array of bytes.

- **createFromIntegerArray**

```
public static Numeric createFromIntegerArray(int b[])
```

Return a Numeric created from the given byte array. The array is considered a string of bits with the most significant bit at the 0th position..

Parameters:

b - An array of bytes.

- **pow**

```
public Numeric pow(int e)
```

Returns this to the e'th power. A binary power algorithm is used.

Returns:

this value to the e'th power

- **pi**

```
public static Numeric pi(int val)
```

Returns pi to the number of places given by the argument. This is good for about 700 places.

- **sqrt**

```
public Numeric sqrt()
```

Returns the square root of this Numeric. Uses a Newtonian convergence algorithm. Not terrifically, efficient.

- **isProbablePrime**

```
public boolean isProbablePrime()
```

This method tests for primality, first by attempting to divide by a few small primes, then by using the Rabin probabilistic primality test 50 time. The probability of failing this test is less than 1 / (4^n).

- **modExp**

```
public Numeric modExp(Numeric a2,
                      Numeric mod)
```

Modular exponentiation

- **significantBits**

```
public int significantBits()
```

  Returns the number of significant bits.

- **shiftRight**

```
public Numeric shiftRight(int sh)
```

  Shift the Numeric right by sh bits.

- **shiftLeft**

```
public Numeric shiftLeft(int sh)
```

  Shift left sh bits.

- **random**

```
public static Numeric random(int bits,
                             Random rnd)
```

- **dump**

```
public void dump(String s)
```

- **toString**

```
public String toString(int radix)
```

  Convert to a String.
  Parameters:
    radix - the radix to be used

- **modInverse**

```
public Numeric modInverse(Numeric q)
```

  Multiplicative inverse k^-1 mod q

Class java.sql.Time

```
java.lang.Object
   |
   +----java.util.Date
           |
           +----java.sql.Time
```

---

public class **Time**
extends Date

This class is used to represent a subset of the standard java.util.date information. We only deal with hours, minutes, and seconds. This lets us represent SQL TIME information.

Constructor Index

- **Time**(int, int, int)
  Construct a Time

Method Index

- **toString**()
  Format a time as hh:mm:ss
- **valueOf**(String)
  Convert a formatted string to a Time value

Constructors

Time

```
public Time(int hour,
            int minute,
            int second)
```

Construct a Time
Parameters:
hour - 0 to 23
minute - 0 to 59
second - 0 to 59

Methods

valueOf

```
public static Time valueOf(String s)
```

Convert a formatted string to a Time value
Parameters:
s - time in format "hh:mm:ss"
Returns:

corresponding Time

- **toString**

```
public String toString()
```

Format a time as hh:mm:ss
Returns:
a formatted time String
Overrides:
toString in class Date

Class java.sql.Timestamp

```
java.lang.Object
   |
   +----java.util.Date
           |
           +----java.sql.Timestamp
```

---

public class **Timestamp**
extends Date

This class extends the standard sun.util.date class with nanos. This lets it represent SQL timestamps.

Note: The granularity of sub-second timestamp precision may vary between database fields, and stored values will get rounded to the field's internal precision.

Constructor Index

- **Timestamp**(int, int, int, int, int, int, int)
  Construct a Timestamp

Method Index

- **equals**(Timestamp)
  Test Timestamp values for equality
- **getNanos**()
  Get the Timestamp's nanos value
- **setNanos**(int)
  Set the Timestamp's nanos value

- **toString**()
  Format a Timestamp as yyyy-mm-dd hh:mm:ss.f...
- **valueOf**(String)
  Convert a formatted string to a Timestamp value

Constructors

Timestamp

```
public Timestamp(int year,
                 int month,
                 int date,
                 int hour,
                 int minute,
                 int second,
                 int nano)
```

Construct a Timestamp
Parameters:
year - year-1900
month - 0 to 11
day - 1 to 31
hour - 0 to 23
minute - 0 to 59
second - 0 to 59
nano - 0 to 999,999,999

Methods

valueOf

```
public static Timestamp valueOf(String s)
```

Convert a formatted string to a Timestamp value
Parameters:
s - timestamp in format "yyyy-mm-dd hh:mm:ss.f"
Returns:
corresponding Date

toString

```
public String toString()
```

Format a Timestamp as yyyy-mm-dd hh:mm:ss.f...
Returns:
a formatted timestamp String
Overrides:
toString in class Date

- **getNanos**

```
public int getNanos()
```

Get the Timestamp's nanos value
Returns:
the Timestamp's fractional seconds component

- **setNanos**

```
public void setNanos(int n)
```

Set the Timestamp's nanos value
Parameters:
n - the new fractional seconds component

- **equals**

```
public boolean equals(Timestamp ts)
```

Test Timestamp values for equality
Parameters:
ts - the Timestamp value to compare with

Class java.sql.Types

```
java.lang.Object
   |
   +----java.sql.Types
```

---

public class **Types**
extends Object

This class defines constants that are used to identify SQL types. The actual type constant values are equivalent to those in XOPEN.

Variable Index

- **BIGINT**
- **BINARY**
- **BIT**
- **CHAR**

- **DATE**
- **DECIMAL**
- **DOUBLE**
- **FLOAT**
- **INTEGER**
- **LONGVARBINARY**
- **LONGVARCHAR**
- **NULL**
- **NUMERIC**
- **OTHER**

  OTHER indicates that the SQL type is database specific and gets mapped to a Java object which can be accessed via getObject and setObject.
- **REAL**
- **SMALLINT**
- **TIME**
- **TIMESTAMP**
- **TINYINT**
- **VARBINARY**
- **VARCHAR**

Constructor Index

- **Types**()

Variables

- **BIT**

  ```
  public final static int BIT
  ```

- **TINYINT**

  ```
  public final static int TINYINT
  ```

- **SMALLINT**

  ```
  public final static int SMALLINT
  ```

- **INTEGER**

  ```
  public final static int INTEGER
  ```

- **BIGINT**

  ```
  public final static int BIGINT
  ```

- **FLOAT**

public final static int FLOAT

- **REAL**

  public final static int REAL

- **DOUBLE**

  public final static int DOUBLE

- **NUMERIC**

  public final static int NUMERIC

- **DECIMAL**

  public final static int DECIMAL

- **CHAR**

  public final static int CHAR

- **VARCHAR**

  public final static int VARCHAR

- **LONGVARCHAR**

  public final static int LONGVARCHAR

- **DATE**

  public final static int DATE

- **TIME**

  public final static int TIME

- **TIMESTAMP**

  public final static int TIMESTAMP

- **BINARY**

  public final static int BINARY

- **VARBINARY**

  public final static int VARBINARY

- **LONGVARBINARY**

  public final static int LONGVARBINARY

- **NULL**

  ```
  public final static int NULL
  ```

- **OTHER**

  ```
  public final static int OTHER
  ```

  OTHER indicates that the SQL type is database specific and gets mapped to a Java object which can be accessed via getObject and setObject.

Constructors

- **Types**

  ```
  public Types()
  ```

Class java.sql.DataTruncation

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.sql.SQLException
                           |
                           +----java.sql.SQLWarning
                                   |
                                   +----java.sql.DataTruncation
```

---

public class **DataTruncation**
extends SQLWarning

When JDBC unexpectedly truncates a data value it reports a DataTruncation warning (on reads) or throws a DataTruncation exception (on writes). The SQLstate for a DataTruncation is set to "01004".

Constructor Index

- **DataTruncation**(int, boolean, boolean, int, int)

Method Index

Method Index

- **getDataSize**()
  Get the number of bytes of data that should have been transferred.
- **getIndex**()
  Get the index of the column or parameter that was truncated.
- **getParameter**()
  True if the value was a parameter; false if it was a column value.
- **getRead**()
  True if the value was truncated when read from the database; false if the data was truncated on a write.
- **getTransferSize**()
  Get the number of bytes of data actually transferred.

Constructors

- **DataTruncation**

```
public DataTruncation(int index,
                      boolean parameter,
                      boolean read,
                      int dataSize,
                      int transferSize)
```

Methods

- **getIndex**

```
public int getIndex()
```

  Get the index of the column or parameter that was truncated.

  This may be -1 if the column or parameter index is unknown, in which case the "parameter" and "read" fields should be ignored.

  Returns:
  the DataTruncation’s index value

- **getParameter**

```
public boolean getParameter()
```

  True if the value was a parameter; false if it was a column value.

  Returns:
  the DataTruncation’s parameter value

- **getRead**

```
public boolean getRead()
```

True if the value was truncated when read from the database; false if the data was truncated on a write.

Returns:

the DataTruncation’s read value

- **getDataSize**

```
public int getDataSize()
```

Get the number of bytes of data that should have been transferred. This number may be approximate if data conversions were being performed. The value may be "-1" if the size is unknown.

Returns:

the DataTruncation’s dataSize value

- **getTransferSize**

```
public int getTransferSize()
```

Get the number of bytes of data actually transferred. The value may be "-1" if the size is unknown.

Returns:

the DataTruncation’s transferSize value

Class java.sql.SQLException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.sql.SQLException
```

---

public class **SQLException**
extends Exception

The SQLException class provides information on a database access error.

Each SQLException provides several kinds of information:

- a string describing the error. This is used as the Java Exception message, and is available via the getMesage() method
- A "SQLstate" string which follows the XOPEN SQLstate conventions. The values of the SQLState string as described in the XOPEN SQL spec.

- An integer error code that is vendor specific. Normally this will be the actual error code returned by the underlying database.
- A chain to a next Exception. This can be used to provided additional error information.

Constructor Index

- **SQLException**()
  Construct an SQLException with no description
- **SQLException**(String)
  Construct an SQLException with only a reason
- **SQLException**(String, String)
  Construct an SQLException without a vendorCode
- **SQLException**(String, String, int)
  Construct a fully specified SQLException

Method Index

- **getErrorCode**()
  Get the vendor specific exception code
- **getNextException**()
  Get the exception chained to this one.
- **getSQLState**()
  Get the SQLState
- **setNextException**(SQLException)
  Add an SQLException to the end of the chain.

Constructors

SQLException

```
public SQLException(String reason,
                    String SQLState,
                    int vendorCode)
```

Construct a fully specified SQLException

Parameters:

reason - a description of the exception
SQLState - an XOPEN code identifying the exception
vendorCode - a database vendor specific exception code

SQLException

```
public SQLException(String reason,
```

```
                    String SQLState)
```

Construct an SQLException without a vendorCode
Parameters:
reason - a description of the exception
SQLState - an XOPEN code identifying the exception

- **SQLException**

```
public SQLException(String reason)
```

Construct an SQLException with only a reason
Parameters:
reason - a description of the exception

- **SQLException**

```
public SQLException()
```

Construct an SQLException with no description

Methods

- **getSQLState**

```
public String getSQLState()
```

Get the SQLState
Returns:
the SQLException's SQLState value

- **getErrorCode**

```
public int getErrorCode()
```

Get the vendor specific exception code
Returns:
the SQLException's vendorCode value

- **getNextException**

```
public SQLException getNextException()
```

Get the exception chained to this one.
Returns:
the next SQLException in the chain

- **setNextException**

```
public synchronized void setNextException(SQLException ex)
```

Add an SQLException to the end of the chain.
Parameters:
ex - the new end of the SQLException chain

Class java.sql.SQLWarning

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.sql.SQLException
                           |
                           +----java.sql.SQLWarning
```

---

public class **SQLWarning**
extends SQLException

The SQLWarning class provides information on a database access warnings. Warnings are silently chained to the object who's method caused it to be reported.

See Also:
getWarnings, getWarnings, getWarnings

Constructor Index

- **SQLWarning**()
  Construct an SQLWarning with no description
- **SQLWarning**(String)
  Construct an SQLWarning with only a reason
- **SQLWarning**(String, String)
  Construct an SQLWarning without a vendorCode
- **SQLWarning**(String, String, int)
  Construct a fully specified SQLWarning

Method Index

- **getNextWarning**()

Get the warning chained to this one

- **setNextWarning**(SQLWarning)
  Add an SQLWarning to the end of the chain.

Constructors

- **SQLWarning**

```
public SQLWarning(String reason,
                  String SQLstate,
                  int vendorCode)
```

Construct a fully specified SQLWarning
Parameters:
reason - a description of the warning
SQLState - an XOPEN code identifying the warning
vendorCode - a database vendor specific warning code

- **SQLWarning**

```
public SQLWarning(String reason,
                  String SQLstate)
```

Construct an SQLWarning without a vendorCode
Parameters:
reason - a description of the warning
SQLState - an XOPEN code identifying the warning

- **SQLWarning**

```
public SQLWarning(String reason)
```

Construct an SQLWarning with only a reason
Parameters:
reason - a description of the warning

- **SQLWarning**

```
public SQLWarning()
```

Construct an SQLWarning with no description

Methods

- **getNextWarning**

```
public SQLWarning getNextWarning()
```

Get the warning chained to this one

Returns:

the next SQLException in the chain

setNextWarning

```
public void setNextWarning(SQLWarning w)
```

Add an SQLWarning to the end of the chain.

Parameters:

w - the new end of the SQLException chain
