java.lang

Class Integer

```
java.lang.Object
  └ java.lang.Number
      └ java.lang.Integer
```

All Implemented Interfaces:
Comparable, Serializable

public final class **Integer**
extends Number
implements Comparable

The `Integer` class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

In addition, this class provides several methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with an `int`.

Since:
JDK1.0
See Also:
Serialized Form

Field Summary

`static int`	**`MAX_VALUE`** A constant holding the maximum value an `int` can have, 2^{31}-1.
`static int`	**`MIN_VALUE`** A constant holding the minimum value an `int` can have, -2^{31}.
`static Class`	**`TYPE`** The `Class` instance representing the primitive type `int`.

Constructor Summary

`Integer``(int value)` Constructs a newly allocated `Integer` object that represents the specified `int` value.
`Integer``(String s)` Constructs a newly allocated `Integer` object that represents the `int` value indicated by the `String` parameter.

Method Summary

`byte`	**`byteValue`**`()` Returns the value of this `Integer` as a `byte`.

int	**compareTo**(Integer anotherInteger) Compares two `Integer` objects numerically.
int	**compareTo**(Object o) Compares this `Integer` object to another object.
static Integer	**decode**(String nm) Decodes a `String` into an `Integer`.
double	**doubleValue**() Returns the value of this `Integer` as a `double`.
boolean	**equals**(Object obj) Compares this object to the specified object.
float	**floatValue**() Returns the value of this `Integer` as a `float`.
static Integer	**getInteger**(String nm) Determines the integer value of the system property with the specified name.
static Integer	**getInteger**(String nm, int val) Determines the integer value of the system property with the specified name.
static Integer	**getInteger**(String nm, Integer val) Returns the integer value of the system property with the specified name.
int	**hashCode**() Returns a hash code for this `Integer`.
int	**intValue**() Returns the value of this `Integer` as an `int`.
long	**longValue**() Returns the value of this `Integer` as a `long`.
static int	**parseInt**(String s) Parses the string argument as a signed decimal integer.
static int	**parseInt**(String s, int radix) Parses the string argument as a signed integer in the radix specified by the second argument.
short	**shortValue**() Returns the value of this `Integer` as a `short`.
static String	**toBinaryString**(int i) Returns a string representation of the integer argument as an unsigned integer in base 2.
static String	**toHexString**(int i) Returns a string representation of the integer argument as an unsigned integer in base 16.
static String	**toOctalString**(int i) Returns a string representation of the integer argument as an unsigned integer in base 8.
String	**toString**() Returns a `String` object representing this `Integer`'s value.
static String	**toString**(int i) Returns a `String` object representing the specified integer.
static String	**toString**(int i, int radix) Returns a string representation of the first argument in the radix specified by the second argument.
static Integer	**valueOf**(String s) Returns an `Integer` object holding the value of the specified `String`.

`static Integer`	`valueOf(String s, int radix)` Returns an `Integer` object holding the value extracted from the specified `String` when parsed with the radix given by the second argument.

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

Field Detail

MIN_VALUE

```
public static final int MIN_VALUE
```

A constant holding the minimum value an `int` can have, -2^{31}.

See Also:
Constant Field Values

MAX_VALUE

```
public static final int MAX_VALUE
```

A constant holding the maximum value an `int` can have, 2^{31}-1.

See Also:
Constant Field Values

TYPE

```
public static final Class TYPE
```

The `Class` instance representing the primitive type `int`.

Since:
JDK1.1

Constructor Detail

Integer

```
public Integer(int value)
```

Constructs a newly allocated `Integer` object that represents the specified `int` value.

Parameters:
`value` - the value to be represented by the `Integer` object.

Integer

```
public Integer(String s)
        throws NumberFormatException
```

Constructs a newly allocated `Integer` object that represents the `int` value indicated by the `String` parameter. The string is converted to an `int` value in exactly the manner used by the `parseInt` method for radix 10.

Parameters:

`s` - the `String` to be converted to an `Integer`.

Throws:

`NumberFormatException` - if the `String` does not contain a parsable integer.

See Also:

`parseInt(java.lang.String, int)`

Method Detail

toString

```
public static String toString(int i,
                              int radix)
```

Returns a string representation of the first argument in the radix specified by the second argument.

If the radix is smaller than `Character.MIN_RADIX` or larger than `Character.MAX_RADIX`, then the radix `10` is used instead.

If the first argument is negative, the first element of the result is the ASCII minus character `'-'` (`'\u002D'`). If the first argument is not negative, no sign character appears in the result.

The remaining characters of the result represent the magnitude of the first argument. If the magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the magnitude will not be the zero character. The following ASCII characters are used as digits:

```
0123456789abcdefghijklmnopqrstuvwxyz
```

These are `'\u0030'` through `'\u0039'` and `'\u0061'` through `'\u007A'`. If `radix` is *N*, then the first *N* of these characters are used as radix-*N* digits in the order shown. Thus, the digits for hexadecimal (radix 16) are `0123456789abcdef`. If uppercase letters are desired, the `String.toUpperCase()` method may be called on the result:

```
Integer.toString(n, 16).toUpperCase()
```

Parameters:

`i` - an integer to be converted to a string.

`radix` - the radix to use in the string representation.

Returns:

a string representation of the argument in the specified radix.

See Also:

`Character.MAX_RADIX`, `Character.MIN_RADIX`

---

toHexString

```
public static String toHexString(int i)
```

Returns a string representation of the integer argument as an unsigned integer in base 16.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise, it is equal to the argument. This value is converted to a string of ASCII digits in hexadecimal (base 16) with no extra leading `0`s. If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The following characters are used as hexadecimal digits:

```
0123456789abcdef
```

These are the characters `'\u0030'` through `'\u0039'` and `'\u0061'` through `'\u0066'`. If uppercase letters are desired, the `String.toUpperCase()` method may be called on the result:

```
Integer.toHexString(n).toUpperCase()
```

Parameters:
`i` - an integer to be converted to a string.
Returns:
the string representation of the unsigned integer value represented by the argument in hexadecimal (base 16).
Since:
JDK1.0.2

---

toOctalString

```
public static String toOctalString(int i)
```

Returns a string representation of the integer argument as an unsigned integer in base 8.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise, it is equal to the argument. This value is converted to a string of ASCII digits in octal (base 8) with no extra leading `0`s.

If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The following characters are used as octal digits:

```
01234567
```

These are the characters `'\u0030'` through `'\u0037'`.

Parameters:
`i` - an integer to be converted to a string.
Returns:
the string representation of the unsigned integer value represented by the argument in octal (base 8).
Since:
JDK1.0.2

---

toBinaryString

```
public static String toBinaryString(int i)
```

Returns a string representation of the integer argument as an unsigned integer in base 2.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in binary (base 2) with no extra leading `0`s. If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The characters `'0'` (`'\u0030'`) and `'1'` (`'\u0031'`) are used as binary digits.

Parameters:
`i` - an integer to be converted to a string.
Returns:
the string representation of the unsigned integer value represented by the argument in binary (base 2).
Since:
JDK1.0.2

---

toString

```
public static String toString(int i)
```

Returns a `String` object representing the specified integer. The argument is converted to signed decimal representation and returned as a string, exactly as if the argument and radix 10 were given as arguments to the `toString(int, int)` method.

Parameters:
`i` - an integer to be converted.
Returns:
a string representation of the argument in base 10.

---

parseInt

```
public static int parseInt(String s,
                           int radix)
                    throws NumberFormatException
```

Parses the string argument as a signed integer in the radix specified by the second argument. The characters in the string must all be digits of the specified radix (as determined by whether `Character.digit(char, int)` returns a nonnegative value), except that the first character may be an ASCII minus sign `'-'` (`'\u002D'`) to indicate a negative value. The resulting integer value is returned.

An exception of type `NumberFormatException` is thrown if any of the following situations occurs:

- The first argument is `null` or is a string of length zero.
- The radix is either smaller than `Character.MIN_RADIX` or larger than `Character.MAX_RADIX`.
- Any character of the string is not a digit of the specified radix, except that the first character may be a minus sign `'-'` (`'\u002D'`) provided that the string is longer than length 1.
- The value represented by the string is not a value of type `int`.

Examples:

```
 parseInt("0", 10) returns 0
 parseInt("473", 10) returns 473
 parseInt("-0", 10) returns 0
```

```
parseInt("-FF", 16) returns -255
parseInt("1100110", 2) returns 102
parseInt("2147483647", 10) returns 2147483647
parseInt("-2147483648", 10) returns -2147483648
parseInt("2147483648", 10) throws a NumberFormatException
parseInt("99", 8) throws a NumberFormatException
parseInt("Kona", 10) throws a NumberFormatException
parseInt("Kona", 27) returns 411787
```

Parameters:

`s` - the `String` containing the integer representation to be parsed

`radix` - the radix to be used while parsing `s`.

Returns:

the integer represented by the string argument in the specified radix.

Throws:

`NumberFormatException` - if the `String` does not contain a parsable `int`.

---

parseInt

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign `'-'` (`'\u002D'`) to indicate a negative value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.

Parameters:

`s` - a `String` containing the `int` representation to be parsed

Returns:

the integer value represented by the argument in decimal.

Throws:

`NumberFormatException` - if the string does not contain a parsable integer.

---

valueOf

```
public static Integer valueOf(String s,
                              int radix)
                       throws NumberFormatException
```

Returns an `Integer` object holding the value extracted from the specified `String` when parsed with the radix given by the second argument. The first argument is interpreted as representing a signed integer in the radix specified by the second argument, exactly as if the arguments were given to the `parseInt(java.lang.String, int)` method. The result is an `Integer` object that represents the integer value specified by the string.

In other words, this method returns an `Integer` object equal to the value of:

```
new Integer(Integer.parseInt(s, radix))
```

Parameters:

`s` - the string to be parsed.

`radix` - the radix to be used in interpreting `s`

Returns:

an `Integer` object holding the value represented by the string argument in the specified radix.

Throws:

`NumberFormatException` - if the `String` does not contain a parsable `int`.

---

valueOf

```
public static Integer valueOf(String s)
                       throws NumberFormatException
```

Returns an `Integer` object holding the value of the specified `String`. The argument is interpreted as representing a signed decimal integer, exactly as if the argument were given to the `parseInt(java.lang.String)` method. The result is an `Integer` object that represents the integer value specified by the string.

In other words, this method returns an `Integer` object equal to the value of:

```
new Integer(Integer.parseInt(s))
```

Parameters:

`s` - the string to be parsed.

Returns:

an `Integer` object holding the value represented by the string argument.

Throws:

`NumberFormatException` - if the string cannot be parsed as an integer.

---

byteValue

```
public byte byteValue()
```

Returns the value of this `Integer` as a `byte`.

Overrides:

`byteValue` in class `Number`

Returns:

the numeric value represented by this object after conversion to type `byte`.

---

shortValue

```
public short shortValue()
```

Returns the value of this `Integer` as a `short`.

Overrides:

`shortValue` in class `Number`

Returns:

the numeric value represented by this object after conversion to type `short`.

---

intValue

```
public int intValue()
```

Returns the value of this `Integer` as an `int`.

Specified by:
`intValue` in class `Number`

Returns:
the numeric value represented by this object after conversion to type `int`.

---

longValue

```
public long longValue()
```

Returns the value of this `Integer` as a `long`.

Specified by:
`longValue` in class `Number`

Returns:
the numeric value represented by this object after conversion to type `long`.

---

floatValue

```
public float floatValue()
```

Returns the value of this `Integer` as a `float`.

Specified by:
`floatValue` in class `Number`

Returns:
the numeric value represented by this object after conversion to type `float`.

---

doubleValue

```
public double doubleValue()
```

Returns the value of this `Integer` as a `double`.

Specified by:
`doubleValue` in class `Number`

Returns:
the numeric value represented by this object after conversion to type `double`.

---

toString

```
public String toString()
```

Returns a `String` object representing this `Integer`'s value. The value is converted to signed decimal representation and returned as a string, exactly as if the integer value were given as an argument to the `toString(int)` method.

Overrides:
`toString` in class `Object`

Returns:
a string representation of the value of this object in base 10.

---

hashCode

```
public int hashCode()
```

Returns a hash code for this `Integer`.

Overrides:
`hashCode` in class `Object`

Returns:
a hash code value for this object, equal to the primitive `int` value represented by this `Integer` object.

See Also:
`Object.equals(java.lang.Object)`, `Hashtable`

---

equals

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

Compares this object to the specified object. The result is `true` if and only if the argument is not `null` and is an `Integer` object that contains the same `int` value as this object.

Overrides:
`equals` in class `Object`

Parameters:
`obj` - the object to compare with.

Returns:
`true` if the objects are the same; `false` otherwise.

See Also:
`Object.hashCode()`, `Hashtable`

---

getInteger

```
public static Integer getInteger(String nm)
```

Determines the integer value of the system property with the specified name.

The first argument is treated as the name of a system property. System properties are accessible through the `System.getProperty(java.lang.String)` method. The string value of this property is then interpreted as an integer value and an `Integer` object representing this value is returned. Details of possible numeric formats can be found with the definition of `getProperty`.

If there is no property with the specified name, if the specified name is empty or `null`, or if the property does not have the correct numeric format, then `null` is returned.

In other words, this method returns an `Integer` object equal to the value of:

```
getInteger(nm, null)
```

Parameters:
`nm` - property name.

Returns:
the `Integer` value of the property.

See Also:
`System.getProperty(java.lang.String)`, `System.getProperty(java.lang.String,`

java.lang.String)

---

getInteger

```
public static Integer getInteger(String nm,
                                 int val)
```

Determines the integer value of the system property with the specified name.

The first argument is treated as the name of a system property. System properties are accessible through the `System.getProperty(java.lang.String)` method. The string value of this property is then interpreted as an integer value and an `Integer` object representing this value is returned. Details of possible numeric formats can be found with the definition of `getProperty`.

The second argument is the default value. An `Integer` object that represents the value of the second argument is returned if there is no property of the specified name, if the property does not have the correct numeric format, or if the specified name is empty or `null`.

In other words, this method returns an `Integer` object equal to the value of:

```
getInteger(nm, new Integer(val))
```

but in practice it may be implemented in a manner such as:

```
 Integer result = getInteger(nm, null);
 return (result == null) ? new Integer(val) : result;
```

to avoid the unnecessary allocation of an `Integer` object when the default value is not needed.

Parameters:

`nm` - property name.

`val` - default value.

Returns:

the `Integer` value of the property.

See Also:

`System.getProperty(java.lang.String)`, `System.getProperty(java.lang.String, java.lang.String)`

---

getInteger

```
public static Integer getInteger(String nm,
                                 Integer val)
```

Returns the integer value of the system property with the specified name. The first argument is treated as the name of a system property. System properties are accessible through the `System.getProperty(java.lang.String)` method. The string value of this property is then interpreted as an integer value, as per the `Integer.decode` method, and an `Integer` object representing this value is returned.

- If the property value begins with the two ASCII characters `0x` or the ASCII character `#`, not followed by a minus sign, then the rest of it is parsed as a hexadecimal integer exactly as by the method `valueOf(java.lang.String, int)` with radix 16.
- If the property value begins with the ASCII character `0` followed by another character, it is parsed as an octal integer exactly as by the method `valueOf(java.lang.String, int)`

with radix 8.
- Otherwise, the property value is parsed as a decimal integer exactly as by the method `valueOf(java.lang.String, int)` with radix 10.

The second argument is the default value. The default value is returned if there is no property of the specified name, if the property does not have the correct numeric format, or if the specified name is empty or `null`.

Parameters:
`nm` - property name.
`val` - default value.

Returns:
the `Integer` value of the property.

See Also:
`System.getProperty(java.lang.String)`, `System.getProperty(java.lang.String, java.lang.String)`, `decode(java.lang.String)`

---

decode

```
public static Integer decode(String nm)
                      throws NumberFormatException
```

Decodes a `String` into an `Integer`. Accepts decimal, hexadecimal, and octal numbers numbers given by the following grammar:

DecodableString:
$Sign_{opt}$ *DecimalNumeral*
$Sign_{opt}$ `0x` *HexDigits*
$Sign_{opt}$ `0X` *HexDigits*
$Sign_{opt}$ `#` *HexDigits*
$Sign_{opt}$ `0` *OctalDigits*

Sign:
`-`

DecimalNumeral, *HexDigits*, and *OctalDigits* are defined in §3.10.1 of the Java Language Specification.

The sequence of characters following an (optional) negative sign and/or radix specifier ("`0x`", "`0X`", "`#`", or leading zero) is parsed as by the `Integer.parseInt` method with the indicated radix (10, 16, or 8). This sequence of characters must represent a positive value or a `NumberFormatException` will be thrown. The result is negated if first character of the specified `String` is the minus sign. No whitespace characters are permitted in the `String`.

Parameters:
`nm` - the `String` to decode.

Returns:
a `Integer` object holding the `int` value represented by `nm`

Throws:
`NumberFormatException` - if the `String` does not contain a parsable integer.

Since:
1.2

See Also:
`parseInt(java.lang.String, int)`

---

compareTo

```
public int compareTo(Integer anotherInteger)
```

Compares two `Integer` objects numerically.

Parameters:
`anotherInteger` - the `Integer` to be compared.

Returns:
the value `0` if this `Integer` is equal to the argument `Integer`; a value less than `0` if this `Integer` is numerically less than the argument `Integer`; and a value greater than `0` if this `Integer` is numerically greater than the argument `Integer` (signed comparison).

Since:
1.2

---

compareTo

```
public int compareTo(Object o)
```

Compares this `Integer` object to another object. If the object is an `Integer`, this function behaves like `compareTo(Integer)`. Otherwise, it throws a `ClassCastException` (as `Integer` objects are only comparable to other `Integer` objects).

Specified by:
`compareTo` in interface `Comparable`

Parameters:
`o` - the `Object` to be compared.

Returns:
the value `0` if the argument is a `Integer` numerically equal to this `Integer`; a value less than `0` if the argument is a `Integer` numerically greater than this `Integer`; and a value greater than `0` if the argument is a `Integer` numerically less than this `Integer`.

Throws:
`ClassCastException` - if the argument is not an `Integer`.

Since:
1.2

See Also:
`Comparable`

Submit a bug or feature
For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.