[*Java*](http://leepoint.net/notes-java/index.html)*: Converting Numbers to Strings*

## Number to string conversion alternatives

* Concatenation (+): Anything concatenated to a string is converted to string (eg, "weight = " + kilograms).
* *java.text.DecimalFormat* gives you precise control over the formating of numbers (number of decimal places, scientific notation, locale formatting, ...).
* Individual wrapper class methods, eg, Integer.toString(*i*). concatenation works as well for the simple cases, but there are some interesting additional conversions here.
* printf(), added in Java 5, uses a format for conversion.
* *No conversion required*. Some common system methods will take any type and convert it, eg,System.out.println().

## Concatenation (+)

The most common idiom to convert something to a string is to concatenate it with a string. If you just want the value with no additional text, concatenate it with the *empty string*, one with no characters in it ("").

If either operand of a concatenation is a string, the other operand is converted to string, regardless of whether it is a primitive or object type.

int x = 42;

String s;

s = x; // ILLEGAL

s = "" + x; // Converts int 42 to String "42"

s = x + " is OK"; // Assigns "42 is OK" to s.

s = "" + 3.5; // Assigns "3.5" to s

s = "" + 1.0/3.0; // Assigns "0.3333333333333333" to s

**Precision**. The problem with floating-point numbers by concatenation is that you have no control over the number of decimal places the conversion chooses for you, as you can see in the above example.

## java.text.DecimalFormat

The *java.text.DecimalFormat* class provides many ways to format numbers into strings, including number of fraction digits, using a currency symbol ($12.35), scientific notation (3.085e24), percentage scaling (33%), locale which defines national formatting options (3,000.50 or 3.000,50 or 3'000,50 or ...), patterns for positive, zero, and negative numbers, etc. These notes show only how to specify the number of fraction digits. Check the Java API documentation for other options.

First, create a DecimalFormat object which specifies the format of the number. The zero before the decimal point means that at least one digit is produced, even if it is zero. The zeros after the decimal point specify how many fraction digits are produced. Then call the *format()* method of that object, passing the numeric value to be converted.

The string that is returned by the *format()* will not contain additional text, other than an optional currency or percent sign. Specifically, it can not be used to pad numbers on the left with blanks for use in right alignment of a column of numbers. Use *printf()* for this.

This program uses the same formatting object many times.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | // File: formatTest/FormatTest.java  // Description: Test default and DecimalFormat formatting.  // Author: Michael Maus  // Date: 2000-10-27, 2000-11-07, 2005-02-13  import java.text.DecimalFormat;  public class FormatTest {  public static void main(String[] args) {  DecimalFormat twoPlaces = new DecimalFormat("0.00");  for (int i=1; i<=10; i++) {  double val = 1.0 / i;  System.out.println("1/" + i + " = " + twoPlaces.format(val) + ", " + val);  }  }  } |

Produces this output. The first floating point number is converted by a 2 decimal place DecimalFormat object and the second is the default conversion.

1/1 = 1.00, 1.0

1/2 = 0.50, 0.5

1/3 = 0.33, 0.3333333333333333 // Round down.

1/4 = 0.25, 0.25

1/5 = 0.20, 0.2

1/6 = 0.17, 0.16666666666666666 // Round up.

1/7 = 0.14, 0.14285714285714285

1/8 = 0.12, 0.125 // Half-even round down.

1/9 = 0.11, 0.1111111111111111

1/10 = 0.10, 0.1

## Rounding by the *half even* algorithm

Floating-point numbers may be *rounded* when they are converted to string (default conversion, DecimalFormator the Math.round method, etc). Rounding uses the *half even* method which rounds

* To the *nearest neighbor*.
* If the number is exactly between its neighbors, it is rounded to its *even neighbor*. The reason middle values are rounded to the even value, instead of up as is commonly done, is to prevent an accumulation of errors.

Examples rounding to an integer: 3.1 rounds to 3, 3.8 rounds to 4, 3.5 rounds to 4, but 4.5 also rounds to 4, 5.5 rounds to 6.

## StringBuffer and StringBuilder append() methods - like concatenation

The most efficient way to build strings from parts is to use *StringBuffer* (all versions, synchronized), or the essential identical *StringBuilder* (Java 5, unsynchronized and therefore slightly faster than StringBuffer). Their many *append()* methods take numbers and perform default conversions to string before appending them to their current value.

StringBuilder sb = new StringBuilder();

int i = 42;

sb.append("Answer = ");

sb.append(i); // Converts i to string and appends to end.

sb.append(", ");

sb.append(1.0/3.0);

String s = s.toString(); // s is "Answer = 42, 0.3333333333333333"

## Numeric objects: Integer, Double, BigDecimal, BigInteger, ...

Concatenation works with all objects, not just numbers. When Java needs to convert an object to a String, it calls the object's toString() method. Because every class (object type) has the class Object as an ancestor, every class inherits Object's **toString()** method. This will do something to generate a string from an object, but it will not always be very useful. Many classes override toString() to do something better. When you write a class whose objects might be converted to a String, write a toString method.

## Various numeric class conversion methods

[This will describe the Integer, etc wrapper class methods, as well as the BigInteger and BigDecimal conversion methods.]

## printf()

Java 5 added the *format()* and *printf()* methods to various classes. These convert number according to a *format string*. There are no notes on it here yet.

NumberFormat: setParseIntegerOnly(boolean value)

/\*Output:   
 \* User's number (US): 1,976.083  
 \* \*/  
  
  
**import**java.text.NumberFormat;  
**import**java.util.Locale;  
  
**public class**MainClass {  
  **public static void**main(String args[]) **throws**Exception {  
    NumberFormat numberFormat = NumberFormat.getInstance();  
    numberFormat.setParseIntegerOnly(**false**);  
    **double**usersNumber = 1976.0826;  
  
    numberFormat = NumberFormat.getNumberInstance(Locale.US);  
    System.out.println("User's number (US): " + numberFormat.format(usersNumber));  
      
  }  
}

NumberFormat: setMinimumIntegerDigits(int newValue)

/\*  
0.0 formats as 000.00  
1.0 formats as 001.00  
3.142857142857143 formats as 003.1429  
100.2345678 formats as 100.2346  
 \*/  
**import**java.text.NumberFormat;  
  
**public class**MainClass {  
  
  **public static void**main(String[] av) {  
    **double**data[] = { 0, 1, 22d / 7, 100.2345678 };  
    NumberFormat **form**= NumberFormat.getInstance();  
  
    // Set it to look like 999.99[99]  
    **form**.setMinimumIntegerDigits(3);  
    **form**.setMinimumFractionDigits(2);  
    **form**.setMaximumFractionDigits(4);  
  
    // Now print using it.  
    **for**(**int**i = 0; i < data.length; i++)  
      System.out.println(data[i] + "\tformats as " + **form**.format(data[i]));  
  }  
  
}

NumberFormat: setMinimumFractionDigits(int newValue)

/\*  
0.0 formats as 000.00  
1.0 formats as 001.00  
3.142857142857143 formats as 003.1429  
100.2345678 formats as 100.2346  
 \*/  
**import**java.text.NumberFormat;  
  
**public class**MainClass {  
  
  **public static void**main(String[] av) {  
    **double**data[] = { 0, 1, 22d / 7, 100.2345678 };  
    NumberFormat **form**= NumberFormat.getInstance();  
  
    // Set it to look like 999.99[99]  
    **form**.setMinimumIntegerDigits(3);  
    **form**.setMinimumFractionDigits(2);  
    **form**.setMaximumFractionDigits(4);  
  
    // Now print using it.  
    **for**(**int**i = 0; i < data.length; i++)  
      System.out.println(data[i] + "\tformats as " + **form**.format(data[i]));  
  }  
  
}

NumberFormat.setMaximumFractionDigits(int newValue)

/\*  
0.0 formats as 000.00  
1.0 formats as 001.00  
3.142857142857143 formats as 003.1429  
100.2345678 formats as 100.2346  
 \*/  
**import**java.text.NumberFormat;  
  
**public class**MainClass {  
  
  **public static void**main(String[] av) {  
    **double**data[] = { 0, 1, 22d / 7, 100.2345678 };  
    NumberFormat **form**= NumberFormat.getInstance();  
  
    // Set it to look like 999.99[99]  
    **form**.setMinimumIntegerDigits(3);  
    **form**.setMinimumFractionDigits(2);  
    **form**.setMaximumFractionDigits(4);  
  
    // Now print using it.  
    **for**(**int**i = 0; i < data.length; i++)  
      System.out.println(data[i] + "\tformats as " + **form**.format(data[i]));  
  }  
  
}

NumberFormat: parse(String source)

/\*  
4096.251 parses as 4096.251 and formats as 4,096.251  
 \*/  
**import**java.text.NumberFormat;  
**import**java.text.ParseException;  
  
**public class**MainClass {  
  
  **public static void**main(String[] av) {  
  
    String input = "4096.251";  
    NumberFormat defForm = NumberFormat.getInstance();  
  
    **try**{  
      Number d = defForm.parse(input);  
      System.out.println(input + " parses as " + d + " and formats as " + defForm.format(d));  
    } **catch**(ParseException pe) {  
      System.err.println(input + "not parseable!");  
    }  
  }  
  
}

NumberFormat: getNumberInstance(Locale inLocale)

/\*Output:   
 \* User's number (US): 1,976.083  
 \* \*/  
  
  
**import**java.text.NumberFormat;  
**import**java.util.Locale;  
  
**public class**MainClass {  
  **public static void**main(String args[]) **throws**Exception {  
    NumberFormat numberFormat = NumberFormat.getInstance();  
    numberFormat.setParseIntegerOnly(**false**);  
    **double**usersNumber = 1976.0826;  
  
    numberFormat = NumberFormat.getNumberInstance(Locale.US);  
    System.out.println("User's number (US): " + numberFormat.format(usersNumber));  
      
  }  
}

NumberFormat: getInstance()

**import**java.awt.FlowLayout;  
**import**java.awt.Font;  
**import**java.text.Format;  
**import**java.text.NumberFormat;  
**import**java.util.Locale;  
  
**import**javax.swing.BoxLayout;  
**import**javax.swing.JFormattedTextField;  
**import**javax.swing.JFrame;  
**import**javax.swing.JLabel;  
**import**javax.swing.JPanel;  
  
**public class**MainClass {  
  **public static void**main(String args[]) **throws**Exception {  
  
    JFrame frame = **new**JFrame("Number Input");  
    frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
    Font font = **new**Font("SansSerif", Font.BOLD, 16);  
  
    JLabel label;  
    JFormattedTextField input;  
    JPanel panel;  
  
    BoxLayout layout = **new**BoxLayout(frame.getContentPane(), BoxLayout.Y\_AXIS);  
    frame.setLayout(layout);  
  
    Format general = NumberFormat.getInstance();  
    label = **new**JLabel("General/Instance:");  
    input = **new**JFormattedTextField(general);  
    input.setValue(2424.50);  
    input.setColumns(20);  
    input.setFont(font);  
    panel = **new**JPanel(**new**FlowLayout(FlowLayout.RIGHT));  
    panel.add(label);  
    panel.add(input);  
    frame.add(panel);  
  
    frame.pack();  
    frame.setVisible(**true**);  
  }  
}

NumberFormat: getCurrencyInstance(Locale.UK)

**import**java.awt.FlowLayout;  
**import**java.awt.Font;  
**import**java.text.Format;  
**import**java.text.NumberFormat;  
**import**java.util.Locale;  
  
**import**javax.swing.BoxLayout;  
**import**javax.swing.JFormattedTextField;  
**import**javax.swing.JFrame;  
**import**javax.swing.JLabel;  
**import**javax.swing.JPanel;  
  
**public class**MainClass {  
  **public static void**main(String args[]) **throws**Exception {  
  
    JFrame frame = **new**JFrame("Number Input");  
    frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
    Font font = **new**Font("SansSerif", Font.BOLD, 16);  
  
    JLabel label;  
    JFormattedTextField input;  
    JPanel panel;  
  
    BoxLayout layout = **new**BoxLayout(frame.getContentPane(), BoxLayout.Y\_AXIS);  
    frame.setLayout(layout);  
  
    Format currency = NumberFormat.getCurrencyInstance(Locale.UK);  
    label = **new**JLabel("UK Currency:");  
    input = **new**JFormattedTextField(currency);  
    input.setValue(2424.50);  
    input.setColumns(20);  
    input.setFont(font);  
    panel = **new**JPanel(**new**FlowLayout(FlowLayout.RIGHT));  
    panel.add(label);  
    panel.add(input);  
    frame.add(panel);  
  
    frame.pack();  
    frame.setVisible(**true**);  
  }  
}

NumberFormat: format(Object obj)

/\*  
4096.251 parses as 4096.251 and formats as 4,096.251  
 \*/  
**import**java.text.NumberFormat;  
**import**java.text.ParseException;  
  
**public class**MainClass {  
  
  **public static void**main(String[] av) {  
  
    String input = "4096.251";  
    NumberFormat defForm = NumberFormat.getInstance();  
  
    **try**{  
      Number d = defForm.parse(input);  
      System.out.println(input + " parses as " + d + " and formats as " + defForm.format(d));  
    } **catch**(ParseException pe) {  
      System.err.println(input + "not parseable!");  
    }  
  }  
  
}

|  |  |  |
| --- | --- | --- |
| |  | | --- | |  | |  | |
| |  | | --- | | NumberFormat.getCurrencyInstance()   **import**java.text.NumberFormat;  **public class**Mortgage {   **public static void**main(String[] args) {     **double**payment = Math.random() \* 1000;     System.out.println("Your payment is ");     NumberFormat nf = NumberFormat.getCurrencyInstance();     System.out.println(nf.format(payment));   } } | |