Formatting currencies in Java

**What’s the problem?** When dealing with money amounts in Java, the primitive *double* type works reasonably well, but has a couple of problems:

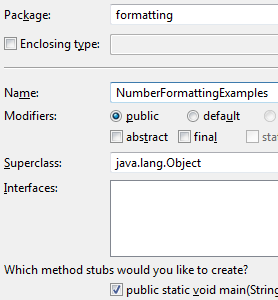
* The double type is based on a binary system, so it can exactly represent fractional numbers that can be expressed as negative powers of two, such as 0.125 (2-3). However, it cannot represent most currency values, which are expressed in positive and negative powers of 10. For example, 34.11 = 3x101+4x100+1x10-11x10-2 cannot be precisely expressed as any combination of powers of 2.

Fortunately, the double type has 53 bits of precision, equivalent to about 16 decimal digits.[[1]](#footnote-1) So the representation of a currency amount as a double will be very close to exact in most applicatioons.

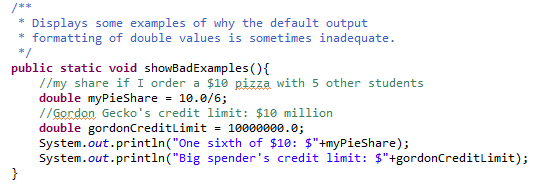
* Also, System.out.println() formats doubles using default rules that often produce ugly results, such as long strings of digits or scientific notation.

For those applications in which “very close” is not good enough, you must use the Java ***BigDecimal*** class. See <http://www.javapractices.com/topic/TopicAction.do?Id=13> for some examples.

This tutorial covers only the formatting issue, which can be solved using the formatting tools in the java.text package.

1. Launch Eclipse and create a new Java Project named Java Examples. You can use this as a collection for all of your example code for exploring new Java features, like this.
2. Create a Java Main class named ***NumberFormattingExamples***, in package formatting. To make this a Main class, check the box for creating a main method.
3. Give your new class a class comment, “Illustrates use of Java number formatting and the BigDecimal class.” The class-level comment uses Javadoc syntax (“/\*\* .. \*/” and immediately precedes the class declaration line.

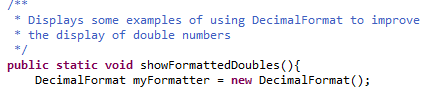
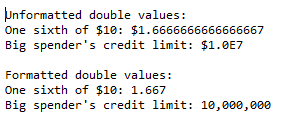
## Bad Examples of number formatting

1. Define a Java method **showBadExamples** as follows:

(Why is this method declared as *static*?)

1. Arrange that this method will be called by the main method and run it. You should see some ugly results!

## Formatting double values

1. Now, look up the DecimalFormat class (just Google for “DecimalFormat Java”). You don’t need to read through the complex examples; just note the two constructors, which either use the default pattern or take a pattern argument.
2. Let’s start by using the default constructor, as shown at right:
3. Continuing the showFormattedDoubles method, add code to calculate the same two local variables, myPieShare and gordonCreditLimit.
4. Look up the method for formatting a double as a String in the DecimalFormat documentation. *Hint*: this method is inherited from the **NumberFormat** class.
5. Complete the showFormattedDoubles method by printing out the formatted result for each number.
6. Call your new method from within your main method. Add labeling printouts so that you can tell the formatted from the unformatted results:
7. Note that DecimalFormat uses the standard rounding rules to round .6666666666667 up to .667.
8. This is an improvement, but we can do better! Replace the default formatter with one created using the pattern string ”000000000.00”. This time, you should see only two digits in the pizza share, but you will get a lot of unwanted leading zeroes:
9. Refer to the “pattern” section of the DecimalFormat Javadoc to find the pattern character that suppresses leading zeroes. Rerun the code to make sure you can suppress leading zeroes.
10. Now insert the decimal separator code “,” into the pattern string, to get the commas.
11. Finally, add the local currency symbol to the beginning of the pattern. Yes, it does start with “\u”! Note that Java uses the “locale” to determine that your currency symbol is a dollar sign. Changing the locale to England will change the dollar sign to £, but won’t convert dollars to British pounds!

1. This requires a bit of thought. Consult <http://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3> for the gory details. Roughly, though, 10 bits equals 3 decimal digits, since 1024 is approximately 1000. So 53 bits is roughly 16 decimal digits. [↑](#footnote-ref-1)