CROSS | OVER

Crossover Technical Trial

TECHNICAL DESIGN DOCUMENT

Number Prettifier

October 25, 2015

Carlos Carrascal Sánchez

1 Revision History

Version		*Changes*	*Author*
#.	*Date*		
1	25/10/2015	Initial version	Carlos Carrascal

2 Index

3 Introduction

This document describes the technical design used to develop an application requested by Crossover for a Technical trial.

The project will be developed using Java programming language, and the resulting product will be a Java class named NumberPrettifier included in package org.crossover.util.

The deliverables for this project will include:

- Technical design document (this document).
- Source code
- A demonstration video
- A readme.txt file with compiling and running information

The project aim is to develop a component to format numbers using a special truncated format with the number followed by a letter, indicating millions, billions or trillions.

The key design points used in this project are as follows:

- Keep it simple. The expected functionality is relatively simple, allowing us to generate a small component.
- Lightweight project using the minimal external dependencies.
- Easy of use, integrate and maintenance.

4 Technical overview

This section will provide general system characteristics, architecture, dependencies, etc.

4.1. System characteristics

The project is developed in Java, using Java OpenJDK, version 7. The rest of components involved in are as follows:

- Java as programming language, using OpenJDK version 7. Oracle Java SE JDK7 or higher is also valid.

  http://www.oracle.com/technetwork/es/java/javase/downloads/jdk7-downloads-1880260.html

  http://openjdk.java.net/install/

- JUnit v4.12 for unit testing.

  https://github.com/junit-team/junit/wiki/Download-and-Install

- Ant v1.9.6 as command line tool used to compile the project.

  http://ant.apache.org/

- Development was made using a Debian 7 GNU / Linux box.

The resulting products are:

- Main class NumberPretiffier, included in package org.crossover.util
- JUnit test classes
- Javadoc documentation

4.2. Dependencies

The only Java dependencies needed for this project outside the JDK Java API is JUnit, and in shipped with the source code of the project.

One of the main goals of the project was to keep dependencies to a minimum, for simplicity's shake.

5 System design

5.1. Files structure

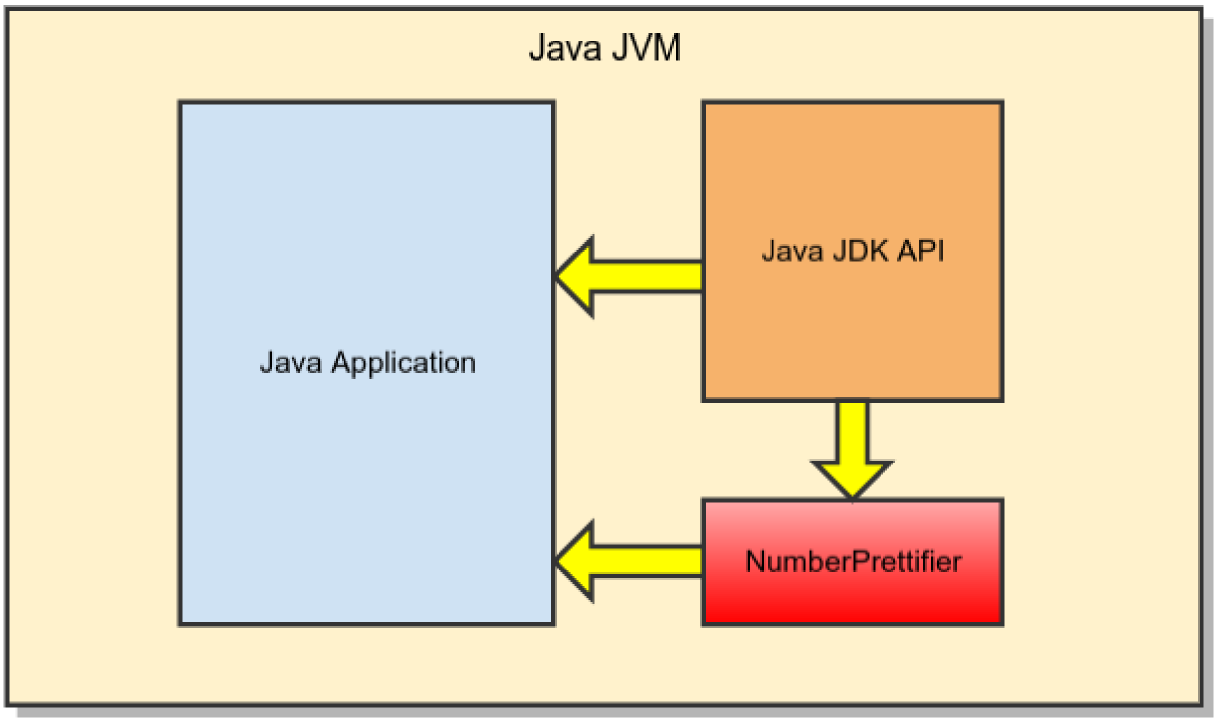
This section describes the files and folder structure used in the project. The following table details all components of the structure.

Folder or file name	Description
bin	Directory containing all Java classes files.
bin/org/crossover/util/	Directory containing class of org.crossover.util package: NumberPrettifier.class
bin/tests/org/crossover/util/	Directory containing testing classes for org.crossover.util package. NumberPrettifierTest.class NumberPrettifierTestSuite.class
docs	Directory containing automatically generated Javadoc files
lib	Directory containing required dependency files, in JAR format. The following files are included: junit-4.12.jar hamcrest-core-1.3.jar
src	Source files directory. ExampleRunner.java : This is an example class that uses NumberPrettifier to display some formatting examples of use.
src/org/crossover/util	Source files for org.crossover.utl package Contain the following files: NumberPrettifier.java
test	Source files for JUnit test classes
test/org/crossover/util	Source files for org.crossover.util package test files.

	NumberPrettifierTest.java NumberPrettifierTestSuite.java
build.xml	Ant file used to compile the projejct, run tests and generate Javadoc files.
compile.sh	Bash shell script for simple compilation without Ant
runtest.sh	Bash shell script for running JUnit tests
run.sh	Bash shell script for running the example. This script executes the class RunningExample, and can receive as a parameter a number to prettify.

5.2. System architecture

The following architecture diagram shows the interaction of this software component with a Java application.



Any Java application can include this component by including the corresponding classes in their Classpath, and referencing the component using an *import* directive.

6 Component description

This section provides a functional overview of the project, as well as detailed information of all the components included in the project.

6.1. Functional overview

The main function of this project is the development of a Java class named *NumberPrettifier*, capable of formatting numbers. This class will be included in a package named *org.crossover.util*.

The class accepts a numeric type as an input and returns a truncated, "prettified" string version. The prettified version includes one number after the decimal when the truncated number is not an integer. It will prettify numbers greater than 6 digits supporting millions (M), billions (B) and trillions (T).

Short scale numbers are used to output the prettified number.

The following examples illustrates the expected output of the component.

```
input: 1000000 output: 1M

input: 2500000.34 output: 2.5M

input: 532 output: 532

input: 1123456789 output: 1.1B

input: 47100000000000 output: 47.1T
```

To deal with decimal representation of the output formats, the JDK class DecimalFormat will be used using the pattern “#.#” to display only one decimal character.

Negative numbers are allowed and formated in the same way.

6.1.1 Regionalization

The *DecimalFormat* class used to display the results will format those results considering the available locales configured in the executing system. Depending on the locale used, the symbol used to represent the decimal separator may change.

In order to obtain consistent output and ensure the results of the unit tests, the component will use US locale by default.

The locale used can be specified in the class constructor, to adapt the results as needed.

As an example, the spanish locale uses the symbol “,” as decimal separator, instead of “.” used by US locale.

6.1.2 Rounding

In order to represent the truncated version of the input number, rounding will be used in the output formatting.

The rounding method to use will be *java.math.RoundingMode.HALF_EVEN*, which is used by default for the *DecimalFormat* class.

As per Java JDK documentation, this rounding method is as follows:

> Rounding mode to round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor.

6.1.3 Error control

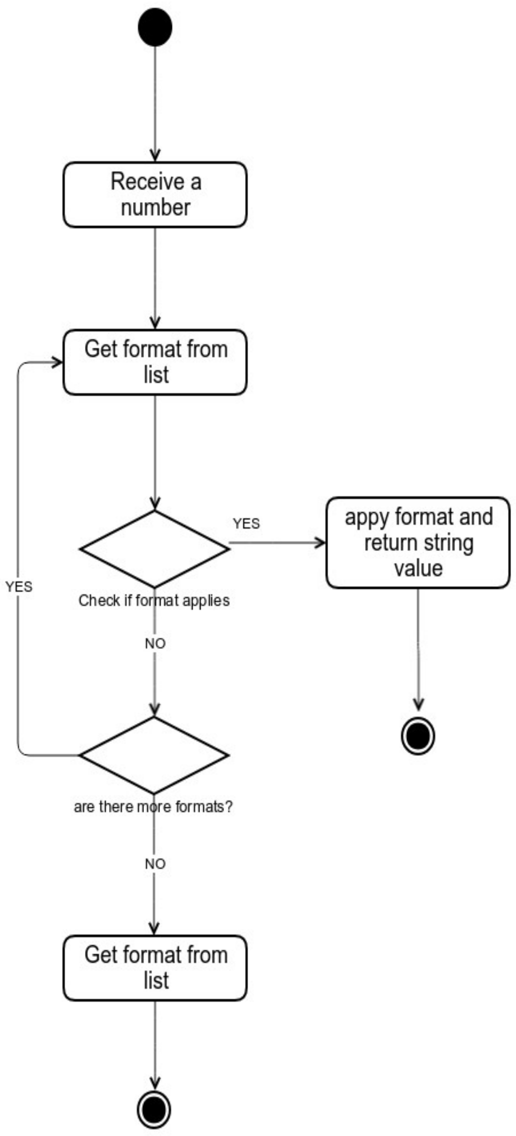
In case of errors during formatting, the class should return an empty value. No exception must be thrown from this class.

This class is expected to be used as a simple formatter. Returning an empty value in case of errors allow the calling class to control the final output without having to check for exceptions or interrupt in the normal execution flow.

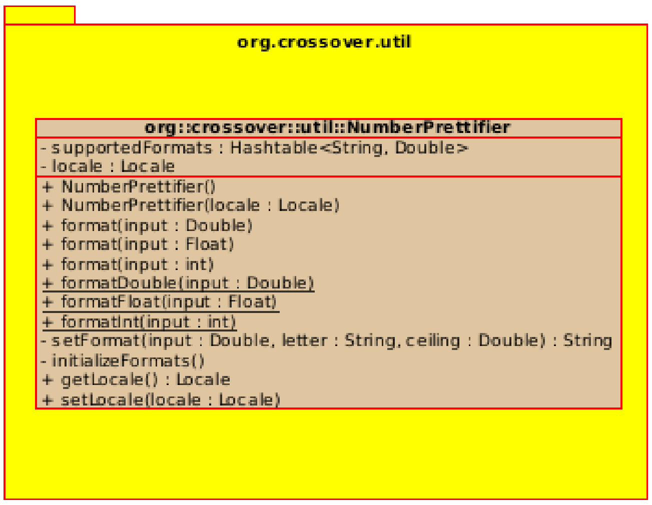
6.2. NumberPrettifier class

This is the main component to be developed in this project. This class will perform the number formatting by using any of the convenience methods to be implemented.

Below is showed the activity diagram of operation of this class. The entry parameter is a number, and depending of the method used, could be an integer, float, double or string value. The output is always a string containing the formatted number, or the original number if no format was applied.



The following image show the diagram for this class, that will be included in the package org.crossover.util.


org.crossover.util
org::crossover::util::NumberPrettifier
- supportedFormats : Hashtable<String, Double>
- locale : Locale
+ NumberPrettifier()
+ NumberPrettifier(locale : Locale)
+ format(input : Double)
+ format(input : Float)
+ format(input : int)
+ formatDouble(input : Double)
+ formatFloat(input : Float)
+ formatInt(input : int)
- setFormat(input : Double, letter : String, ceiling : Double) : String
- initializeFormats()
+ getLocale() : Locale
+ setLocale(locale : Locale)

Class diagram

The class attributes and methods are as follows.

Attributes	**Type**	**Description**
locale	Locale	Private. Locale used to format the output numbers. Used to determine the decimal separator character.
supportedFormats	LinkedHashMap	Private. Map containing pairs of key / value representing the formats to use. The key will be the letter used to append to the result. The value will be the ceiling value to determine when to apply this format. An input number greater or equal to the ceiling value will be formatted with it. The formats have to be introduced in decreasing order to be processed correctly.

Method	Parameters	Return value	Description
NumberPrettifier	-	void	Empty constructor
NumberPrettifier	locale - Locale	void	Constructor with provided *Locale*
InitializeFormats	-	void	Initialize *supportedFormats* map with predefined values.
format	input - Double	String	Public. Main formating method. Format a number into a truncated, "prettified" string version. Negative numbers are allowed and formated in the same way.
setFormat	input – Double letter – String ceiling – Double	String	Private. This function truncates the number as determined by *ceiling* parameter, and uses a *DecimalFormat* class to format the number if the resulting number is not an integer, to represent it only with one decimal. The decimal separator will be "." by default, unless a specific *Locale* was used in the constructor.
format	input - String	String	Public. Helper function to format using a *String*. Calls format(*Double*) internally.
format	input - int	String	Public. Helper function to format using an int. Calls format(*Double*) internally.
formatDouble	input - Double	String	Public, static. This method does the same as format(*Double*) but can be called statically.
formatFloat	input - Float	String	Public, static. This method does the same as format(*Double*) but can be called statically.
formatInt	input - int	String	Public, static. This method does the same as format(*Double*) but can be called statically.

6.3. NumberPrettifierTest JUnit Test class

A JUnit test class will be developed along with the main *NumberPrettifier* class. The tests implemented in this class will be included in the Ant file, so they can be executed automatically each time that the project is compiled.

This class will be included in the same *org.crossover.package* than the tested class, but will be placed in a separate “test” source directory.

This class will implement the following tests.

Test	Description
Basic test	Basic testing of the values provided as examples.
Negative values test	Same test for negative values
Different locale test	Same values with different locale
Wrong locale name test	With a incorrect locale, values must revert to US
Wrong values	Wrong string values should return empty string
Static method test	Test static methods with same values

6.4. Ant build file (build.xml)

An Ant file must be included in the project to control basic tasks:

- Compile the source code.
- Execute unit tests.
- Generate Javadoc from source files, into “doc” directory.

The default action will be compile, test and generate Javadoc.