A Metaprogramming Pattern for Creating Java Class Functions Using a Spreadsheet

Robin M. Snyder
snyderr@savstate.edu

Follow this and additional works at: http://aisel.aisnet.org/sais2006

Recommended Citation
http://aisel.aisnet.org/sais2006/23

This material is brought to you by the Southern (SAIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in SAIS 2006 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
A METAPROGRAMMING PATTERN FOR CREATING JAVA CLASS FUNCTIONS USING A SPREADSHEET

Robin M. Snyder
College of Business Administration, Savannah State University
snyderr@savstate.edu

Abstract

This paper shows how spreadsheet software such as Microsoft Excel can be used as a pattern to create Java program code that forms part of a Java class. Such a programming technique can be used in an introductory programming course to emphasize to the student the concept of a program as text. Although useful in its own right, this metaprogramming technique of creating program code is useful in demonstrating that program code need not be written manually. A complete simple example is provided and important details of a nontrivial example are discussed. Java is used as the target code, but the method can be used with any programming language and is simple enough to be used by programmers at any level. Extensions to some nontrivial statistical problems/functions are briefly discussed.

Keywords: Metaprogramming, spreadsheets, functions

Introduction

Over the past twenty years or so, spreadsheets have become ubiquitous in many quantitative problem solving areas. The use of spreadsheet software has allowed non-programmers to create models and solve problems in areas that previously required extensive manual effort by professional programmers and/or computer scientists, the use of custom (and often inflexible) software packages, or programming ability in the domain specific area of the problems being solved. On the other hand, many programming courses tend to avoid the use of spreadsheet software in favor of explicit programming. From a programming language perspective, it is interesting to note that a spreadsheet model (without calls to external program code) is a simple functional language consisting of functional equations without recursion (i.e., without cycles). In this paper, a spreadsheet is used to create the Java statements to implement a function by table lookup (although other ways are possible). This is a metaprogramming technique as a software system is used to automate the generation of program code. A statistical function example is used, but the method, as presented, is applicable to modeling any simple function. Note, for example, that general-purpose automatic code generation is not easily modeled by a simple function.

To put the problem in perspective, there are times when the needed functionality for a program piece is not easily written in the programming language being used. One option is to find a piece of code that does what is needed (i.e., code reuse). Another option is to learn the required theory and develop the code in-house (or contract it out). For the purposes of this paper, the required functionality is assumed to be readily available in existing spreadsheet software. It is also assumed that there is no convenient electronic interface to the spreadsheet software. For example, a Java applet is being developed and it cannot be assumed that the necessary spreadsheet and interface will be available on the client machine.

Given these assumptions, this paper shows how a spreadsheet model can be used to generate program code to supply nontrivial functionality without coding the details of the functionality (i.e., using that which is available in the spreadsheet software). First, a simple example is shown completely in order to demonstrate the method in detail. Then comments are made on more involved examples.

The method outlined here has been used in a first or second programming course for information systems majors, allowing such students to accomplish something that they could not otherwise do (i.e., using explicit programming). While computer science majors might be tempted to dig into the theory and develop the necessary
code, information systems majors are usually satisfied to find a suitable solution, solve the problem, and move on. The interesting aspect here is that the information system approach is often more cost effective, in a software development sense, than the computer science approach. Using Excel to develop/generate the needed code actually reuses functionality that is built into Excel, but not directly available in or otherwise easily accessible from the programming language being used (e.g., Java).

**Java Code Example**

The idea of the method is to create the necessary functionality in the spreadsheet, create the Java program code in the spreadsheet, and transfer the generated program code from the spreadsheet to a Java class (e.g., via copy-paste) where it can be used.

A simple spreadsheet example to show the method is now presented. Suppose that a Java class is to be developed that provides a function to return the Excel spreadsheet equivalent of the NORMSINV function, a function that is simple but not nontrivial, although the actual use of the NORMSINV function is beyond the scope of the paper (but would be covered in any introductory statistics course). In the NORMSINV function, given a fractional value from 0.1 to 0.9 (values 0.0 and 1.0 are undefined), the corresponding z-value is to be returned. Note that, in practice, many more values might be used. Also keep in mind that the function used here is fairly simple. Real problems might involve much more complex and/or customized functions. The values from 0.1 to 0.9 will be mapped to the index range of 0 to 8. Here is one way to represent the Java class to return any of these z-values using an array. The development of an actual Java function (rather than an array) is left as an exercise in modifying the Java code and/or the Excel spreadsheet.

```java
class normals {
    public static final int count = 9;
    public double normsinv[];
    public normals() {
        normsinv = new double[count];
        normsinv[0] = -1.28155;
        normsinv[1] = -0.84162;
        normsinv[2] = -0.52440;
        normsinv[3] = -0.25335;
        normsinv[4] = 0.00000;
        normsinv[5] = 0.25335;
        normsinv[6] = 0.52440;
        normsinv[7] = 0.84162;
        normsinv[8] = 1.28155;
    }
}
```

Note that this Java class code will be the output of the spreadsheet pattern to be developed.
Given the above Java class code, here is a simple test program for the class that uses the class to output the z-values to the standard output.

```java
import java.text.NumberFormat;
import java.text.DecimalFormat;
public class normal1 {
    public static void main(String [] args) {
        new normal1();
    }
    public normal1() {
        normals ns = new normals();
        NumberFormat f = new DecimalFormat("##.#####");
        for (int i = 0; i < normals.count; i++) {
            System.out.println("normsinv[" + i + "] = " + f.format(ns.normsinv[i]) + ";");
        }
    }
}
```

Here is the output of the simple test program.

```
normsinv[0] = -1.28155;
normsinv[1] = -0.84162;
normsinv[2] = -0.5244;
normsinv[3] = -0.25335;
normsinv[4] = 0;
normsinv[5] = 0.25335;
normsinv[6] = 0.5244;
normsinv[7] = 0.84162;
normsinv[8] = 1.28155;
```

This text output comprises the primary functionality of the developed class. Assuming that the functionality is not so easy to implement, a spreadsheet model could be used to supply the missing functionality (i.e., the above assignment statements). This is now discussed.

**Spreadsheet Model**

Since the spreadsheet will serve as a pattern for a function that takes parameters and returns a value, the class name, function (i.e., array) name, and type returned by the function (i.e., array) are put at the top of the spreadsheet so that they can be easily changed. The class wrapper is created using string expressions. The domain index and function range value results are calculated using the power of the spreadsheet. Here are the spreadsheet formulas and values.
For those not familiar with spreadsheets, the following is an important point. To create the Java code in block C7:C22 from this spreadsheet pattern, or to customize it for another function, two things need to be done changed. First, the class name, function name, and function type is placed into the corresponding cells in block B1:B3. Next, the base case for the range result is placed into cell B12. The base formula in cell B12 is then copied and pasted into block B12:B20. The Java class code is then in block C7:C22 and can be copied and pasted into the Java program where needed.

Note that if the domain count (i.e., number of rows in the domain) is changed, the formula in cell B4 should be changed appropriately. For example, if additional rows are inserted within the block A13:A20, then the formula in cell B4 will be updated automatically. If gaps result, the formula in cell A13 should be copied and pasted into block A13:A20 and the formula in cell B12 should be copied and pasted into block B12:B20. (This is normal spreadsheet practice when inserting rows and/or columns in the middle of a sequence of similar formulas).

Discussion of More Involved Examples

Some design and implementation considerations for some more involved examples are now discussed using the above simple example as a blueprint for how to create the needed functionality when the functionality is not simple. Note that one common limitation of the method presented is that the arguments tend to come from a finite domain. For example, if the standard Z-value is needed for the cumulative standard normal distribution, then one approach is to take the percentages from 1.0 to 99.9 (argument 0.0 would return -infinity while argument 100.0 would return +infinity; this is best handled using standard programming techniques involving conditional statements) and convert them to integers to form a finite domain from 1 to 99 (or 0 to 98). Thus, instead of implementing the code to compute the required normal distribution, the existing functionality of the spreadsheet is used. In general, when real number approximations are returned, the spreadsheet can be used to supply the desired precision and number format.

Most of the nontrivial examples using built-in functions such as NORMSINV, NORMSDIST, FINV, FDIST, CHIDIST, CHINV, POISSON, BINOMDIST, etc., were encountered when students in the author’s Java class were required to implement the solution to statistical problems that the author developed for an introductory statistics class. The online
examples (about 40 classes of problems were created for the statistics) provided multiple problem instances for each
class. Each problem instance included a "Show me how" feature. From this collection of problems, the project
groups were to implement, in Java, an assigned problem class. All problems could be easily couched in terms of a
finite domain of input values resulting in a (functionally determined) output value. In implementation terms, the use
of a listbox to allow the user to select the input values was usually the best choice as the finite domain of input
values could then be easily be specified by the user of the program.

Some functions do not have a complete argument domain. For example, the cumulative binomial distribution
(Poisson, etc.) has three parameters, the probability $p$ (a finite domain of probabilities is used), the number of trials
$n$, and the number of successes $i$ which ranges from 0 to $n$. Creating the spreadsheet model is easier if one model is
created and then copied to each value of $n$. In general, a sophisticated mapping of finite domain arguments to a one-
dimensional array could be used, or a less efficient use of memory could be used with multidimensional arrays with
some index values not used. The conditional spreadsheet function can be used to handle finite domains with
incomplete argument domains. The conditional function has the following form.

$$\text{IF}( \text{expression} , \text{valueIfTrue} , \text{valueIfFalse} )$$

The empty string can be used to not generate code for that combination of values. For space reasons, other design
considerations and implementation details are omitted, but follow from the above methods. In the case of the
binomial example, 13 probability values to $n$ value of 7 resulted in over 500 lines of assignment statements. But, the
program statements can be done quickly, can be easily verified to work correctly (if the spreadsheet software works
correctly), and solves the desired problem.

Summary

This paper has discussed the use of spreadsheet software to generate program code instructions for functionality that
might be difficult to develop in the programming language being used, but might be done easy in a spreadsheet
system. The method is useful for teaching metaprogramming techniques, introducing students to spreadsheet
software in a programming setting, and as another useful tool in the software development process.

References
