Submission Title
Mathematical Function Objects

Contact Information
Richard Rasala rasala@ccs.neu.edu
Viera K. Proulx vkp@ccs.neu.edu
College of Computer & Information Science
Northeastern University
Boston MA 02115

Problem Statement
The encapsulation of functions as objects is a major theme in the Java Power Tools and in its demo programs. In the case of mathematical functions, such encapsulation is critical for enabling the definition of generic methods that apply to all functions. In this submission, we will present a general way to solve this problem.

Solution Overview
This submission will discuss three classes Function, Parameter, and DataTables2D that are currently used in demo programs and will be added to JPT in the next release.

The purpose of the Function and Parameter classes is to collect a family of related interfaces that define what it means to have a mathematical function object with specified properties. We give the complete definition of the Function class:

```java
import java.awt.geom.*;

public class Function {
    public interface NoArg {
        public double evaluate();
    }

    public interface OneArg {
        public double evaluate(double x);
    }

    public interface TwoArg {
        public double evaluate(double x, double y);
    }

    public interface ThreeArg {
        public double evaluate(double x, double y, double z);
    }

    public interface ArrayArg {
        public double evaluate(double[] args);
    }

    public interface TwoArrayArg {
        public double evaluate(double[] args1, double[] args2);
    }

    public interface ArrayOfArrayArg {
        public double evaluate(double[][] args);
    }
}
```

The Parameter class is similar except that it focuses on the auxiliary parameters that define a function. Hence, this class contains interfaces OneParam, TwoParam, ....

Example 1: Elementary Functions
In high school mathematics and in calculus, students study functions of the form \( f(x) \), that is, functions of one variable. In Java terms, we say that these functions implement the interface Function.OneArg.

Example 2: Polynomials
A polynomial in one variable \( x \) has a sequence of coefficients, one for each power of \( x \). Hence, in Java terms, a polynomial implements both Function.OneArg and Parameter.ArrayParam. Below is the heart of the definition of a Polynomial class based on these ideas:

```java
public class Polynomial implements Function.OneArg, Parameter.ArrayParam {
    protected double[] coefficients = null;

    public double evaluate(double x) {
        if (coefficients == null) return 0;

        int d = coefficients.length - 1;
        double result = coefficients[d];
        for (int i = d - 1; i >= 0; i--) {
            result *= x;
            result += coefficients[i];
        }
        return result;
    }

    public void setParam(double[] params) {
        if (params == null) {
            coefficients = null;
            return;
        }

        int d = params.length;
        coefficients = new double[d];
        for (int i = 0; i < d; i++)
            coefficients[i] = params[i];
    }
}
```

In practice, the Polynomial class would have several constructors and a number of convenience methods that we omit here. Notice that we have chosen in this example to use an internal array to hold the coefficient data. With more subtle code, we could use some form of sparse representation.
Example 3: The Big-O functions.

In the submission on PlotTool and PlotMark, we described the Big-O demo program. This program examines functions that are positive constant multiples of the standard functions \( \log(x) \), \( x \), \( \ldots \). We will now see how these Big-O functions are handled.

We first define an abstract class that specifies what we mean by a Big-O function.

```java
public abstract class BigOFunction
    implements Function.OneArg, Parameter.OneParam {
    protected double coefficient = 1;

    public abstract double evaluate(double x);

    public final void setParam(double a) {
        if (a > 0) coefficient = a;
    }
}
```

The Big-O demo then defines a specific BigOFunction for each of the functions that will be tested. Here are two examples:

```java
public static final BigOFunction logx_fcn =
    new BigOFunction() {
    public double evaluate(double x) {
        return coefficient * Math.log(x);
    }
};

public static final BigOFunction x3_logx_fcn =
    new BigOFunction() {
    public double evaluate(double x) {
        return coefficient * x * x * x * Math.log(x);
    }
};
```

Notice that, as in the submission on Actions, we use \{ \} to provide the definition of a missing abstract method. We are able to access the coefficient in the base class since it has protected access.

We now discuss the DataTables2D class whose purpose is to compute tables of function data points to be used in conjunction with PlotTool and PlotMark or for data analysis. This class provides static methods that do the mathematics. Here is one example of its methods:

```java
public static Point2D[] makeTable
    (Function.OneArg f, // the function to evaluate
     double a,        // an interval endpoint
     double b,        // an interval endpoint
     int divisions)   // the subdivisions
{
    if (f == null) return null;

    if (divisions < 1) divisions = 1;

    double x, y;
    double delta = (b - a) / divisions;

    Point2D[] data = new Point2D[divisions + 1];
    for (int i = 0; i <= divisions; i++) {
        if (i < divisions)
            x = a + i * delta;
        else
            x = b; // avoids round off error

        y = f.evaluate(x);

        data[i] = new Point2D.Double(x, y);
    }
    return data;
}
```

The makeTables method illustrates the fact that there is a large gain in flexibility obtained from passing an argument that meets an interface such as Function.OneArg rather than demanding that the argument derive from a specific base class. We believe that it is often very important to use an interface driven design that specifies behavior without a commitment to implementation. This example is designed to show students the benefit of the interface driven design style as compared to base class inheritance style.

Experience with the Solution

This submission and the companion submission on PlotTool and PlotMark really push the envelope on abstraction and on the integration of a functional, behavior-centric style with traditional object-oriented style. We believe students must eventually be able to perform such an integration if they are to become really powerful programmers. This focus on behavior and on treating functions as objects is the essence of the higher-order message from Alan Kay’s talk on the history of Smalltalk.

Nevertheless, we are aware that even some faculty find these ideas difficult. We are hopeful, however, that if students are given the chance to learn generic code such as in the makeTables example above then they will see how powerful these abstraction techniques can be.

API Documentation & Related Materials

The main JPT site to access documentation, code, and the jpt.jar:
http://www.ccs.neu.edu/jpt/

The talk of Alan Kay on Smalltalk may be found at: