Educational Offering

Design Pattern Primer

Exercise: Singleton

Overview
In this exercise, you will apply the Singleton pattern to the calculator factory implementation.

User Requirement
You need to utilize Java compiler and byte code interpreter to create and compile classes, methods, and fields.

Introduction
A previous exercise defined how the AbstractFactory pattern is utilized to produce calculator instances.

Utilizing the factory required an instance to be created; calculator instances were obtained by exercising desired instance methods. The Singleton pattern allows global instances of factories to be accessed eliminating the need for clients having to create and reference an instance.

Source for this exercise is defined in the db.lab.singleton package.

Exercise Instructions
1. Apply the Singleton pattern
In the db.lab.singleton package add the methods shown below to the CalculatorFactory class. Notice the comment. You will also have to add a static attribute with the name of instance that is typed as a CalculatorFactory to the class definition.

```
public static CalculatorFactory getInstance() {

            if (instance == null)
             {  // Concrete class can determined at runtime
                  // by externalizing class name and using
                  // reflection
                  instance =  new ConcreteCalculatorFactory(); }

             return instance;
      }
```

All calculator instances are obtained from a single instance of the ConcreteCalculatorFactory. The testing class below obtains calculators using the Singleton concrete factory.

```
public class Tester {

      public static void main(String[] args) {

            // basic calculator

            System.out.println("* * Basic Calculator * *");

            Calculator calc = CalculatorFactory.getInstance().basic();
            // install operations

            calc.execute("+", 10.0);
            calc.print();
            calc.execute("+", 10.0);
            calc.print();
            calc.execute("-", 10.0);
            calc.print();
            calc.execute("/", 2.0);
            calc.print();
            // Scientific Calculator

            calc = CalculatorFactory.getInstance().scientific();

            System.out.println("* * Scientific Calculator * *");

            //calc.log(10.0)
            calc.execute("+", 10.0);
            calc.execute("sin", 20.0);
            calc.print();
            calc.execute("log", 100);
            calc.print();

            // Programmer Calculator

            calc = CalculatorFactory.getInstance().programmer();

            System.out.println("* * Programmers Calculator * *");

            //calc.log(10.0)
            calc.execute("+", 10.0);
            calc.execute("+", 20.0);
            calc.execute("bin");
            calc.print();

      }

}
```