Educational Offering

Design Pattern Primer

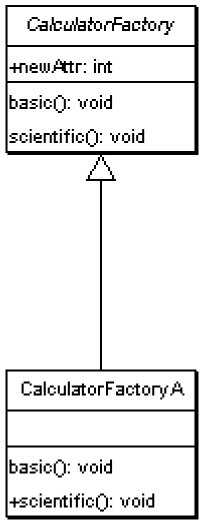
Exercise: Abstract Factory

Overview
In this exercise, you will create different types of calculators using the *AbstractoryFactory* pattern.

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

Introduction
This exercise will describe how the *AbstractFactory* pattern can be applied to produce calculator instances that support a specific grouping of operations. An abstract factory class will be implemented that defines an interface for basic and scientific implementations. A concrete factory will be defined that applies behavior for a concrete implementation

UML for the basic design is shown below:



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

Exercise Instructions

1. Create Abstract Factory class
In the *db.lab*.factory package implement the abstract factory class definition shown below:

```
import dp.lab.strategy.*;
public abstract class CalculatorFactory {

      public abstract Calculator basic();
      public abstract Calculator scientific();

}
```

2. Create a concrete factory implementation by extending the *CalculatorFactory* with a class named *CalculatorFactoryA*. Implement the method definitions below:

```
      public Calculator basic() {

            // basic calculator

            Calculator calc = new Calculator();
            // install operations

            calc.install(new AddOperation());
            calc.install(new SubtractOperation());
            calc.install(new MultiplyOperation());
            calc.install(new DivideOperation());

            return calc;
      }

      public Calculator scientific() {

            // produce scientific calculator

            Calculator calc = new Calculator();

            calc.install(new AddOperation());
            calc.install(new SubtractOperation());
            calc.install(new MultiplyOperation());
            calc.install(new DivideOperation());
            calc.install(new SinOperation());
            calc.install(new TanOperation());
            calc.install(new LogOperation());

            return calc;
      }
```

3. Implement Test Harness Class
Implement and execute the test harness class shown below:

```
import dp.lab.strategy.*;

public class Tester {

      public static void main(String[] args) {
            // create factory

            CalculatorFactory factory = new CalculatorFactoryA();

            // basic calculator

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

            Calculator calc = factory.basic();
            // install operations

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

            calc = factory.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();

      }

}
```