
Educational Offering

# Design Pattern Primer

## Exercise: Calculator Types Using Inheritance

## Overview

In this exercise, you will create additional calculator types using inheritance.

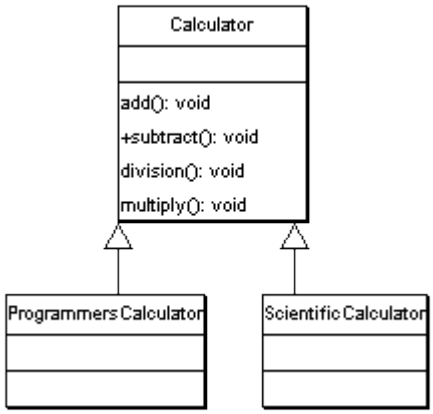
## User Requirement

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

## Introduction

A calculator class exists that implements basic arithmetic functionality. This exercise will implement different types of calculators by extending the Calculator class.

UML for the basic design is shown below:




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

## Exercise Instructions:

**1. Create test harness for the basic Calculator class**

In the ***db.lab*** package, implement a test class shown below. Executing this classes main method should output results to the console. Study the implementation.

```
public class Tester {

      public static void main(String[] args) {
```


*Copyright © 2014 Keyhole Software, LLC. All rights reserved. Tel 877-521-7769 https://keyholesoftware.com*

```
            // basic calculator
            System.out.println("* * Basic Calculator * *");

            Calculator calc = new Calculator();
            calc.add(10.0)
                .add(10.0)
                .subtract(20.0)
                .add(10)
                .multiply(20.22)
                .print();                }}
```

**2. Create a scientific calculator**

Implement a scientific calculator by defining a class named *ScientificCalculator* that extends from *Calculator*. Implement the methods shown below:

```
public Calculator sin(double value) {

            setOperation("sin");
            swap(value);
            setResult( Math.sin(value) );
            return this;

      }

public Calculator log(double value) {

            setOperation("log");
            swap(value);
            setResult( Math.log(value) );
            return this;

      }
```

3. Modify the test class so that the *ScientificCalculator* with testing expressions shown below. Then execute.

```
// Scientific Calculator

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


ScientificCalculator scalc = new ScientificCalculator();
scalc.log(10.0)
.add(10.0);
scalc.sin(20.0);
scalc.print();
```

**4. Create a Programmers Calculator**


*Copyright © 2014 Keyhole Software, LLC. All rights reserved. Tel 877-521-7769 https://keyholesoftware.com*

Implement a programmers calculator by defining a class named *ProgrammersCalculator* that extends from *Calculator*. This calculator implementation provides functionality to output results in binary, hex, and octal radixes. Therefore some attributes are required along with additional methods. The complete implementation is shown below:

```
public class ProgrammerCalculator extends Calculator {

	// display state

	private int displayState = 0;

	// states
	static final int DECIMAL = 0;
	static final int BINARY = 1;
	static final int HEX = 2;
	static final int OCTAL = 3;

	public void bin() {
		setDisplayState(BINARY);
		print();

	}

	public void hex() {
		setDisplayState(HEX);
		print();
	}

	public void octal() {
		setDisplayState(OCTAL);
		print();

	}

	public void decimal() {
		setDisplayState(DECIMAL);
		print();

	}

	/**
	 * Convert expression to binary
	 */

	public String printBinary() {

		int itemp = new Integer((int) getLeftValue()).intValue();
		String sresult = "(bin) " + Integer.toBinaryString(itemp);
		sresult += getOperation();
		itemp = new Integer((int) getRightValue()).intValue();
		sresult += Integer.toBinaryString(itemp);
		sresult += " = ";
		itemp = new Integer((int) getResult()).intValue();
		sresult += Integer.toBinaryString(itemp);
```


*Copyright © 2014 Keyhole Software, LLC. All rights reserved. Tel 877-521-7769 https://keyholesoftware.com*

```
        return sresult;

    }

    /**
     * Convert expression to Hex
     */

    public String printHex() {

        int itemp = new Integer((int) getLeftValue()).intValue();
        String sresult = "(hex) "+Integer.toHexString(itemp);
        sresult += getOperation();
        itemp = new Integer((int) getRightValue()).intValue();
        sresult += Integer.toHexString(itemp);
        sresult += " = ";
        itemp = new Integer((int) getResult()).intValue();
        sresult += Integer.toHexString(itemp);

        return sresult;

    }

    /**
     * Convert expression to binary
     */

    public String printOctal() {

        int itemp = new Integer((int) getLeftValue()).intValue();
        String sresult = "(octal) " + Integer.toOctalString(itemp);
        sresult += getOperation();
        itemp = new Integer((int) getRightValue()).intValue();
        sresult += Integer.toOctalString(itemp);
        sresult += " = ";
        itemp = new Integer((int) getResult()).intValue();
        sresult += Integer.toOctalString(itemp);

        return sresult;

    }

    /**
     * Convert expression to binary
     */

    public String printDecimal() {

        String sresult =
              "(decimal) "
                    + getLeftValue()
                    + " "
                    + getOperation()
                    + " "
                    + getRightValue()
```


*Copyright © 2014 Keyhole Software, LLC. All rights reserved. Tel 877-521-7769 https://keyholesoftware.com*

```
                        + " = "
                        + getResult();

            return sresult;

      }

      public void print() {

            String display = "";
            switch (displayState) {
                  case BINARY :
                        display = printBinary();
                        break;
                  case HEX :
                        display = printHex();
                        break;
                  case OCTAL :
                        display = printOctal();
                        break;
                  case DECIMAL :
                        display = printDecimal();
                        break;

            }

            super.print(display);
      }


      public int getDisplayState() {
            return displayState;
      }


      public void setDisplayState(int displayState) {
            this.displayState = displayState;
      }

}
```

5. Modify the test harness class to include expressions shown below that will exercise the Programmers calculator.

```
      // Programmers Calculator

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

            ProgrammerCalculator pcalc = new ProgrammerCalculator();
            pcalc.add(10.00);
            pcalc.add(20.00);
            pcalc.bin();
            pcalc.octal();
            pcalc.hex();
            pcalc.decimal();
```


*Copyright © 2014 Keyhole Software, LLC. All rights reserved. Tel 877-521-7769 https://keyholesoftware.com*