using System;

namespace BehaviouralPatternDemos

{

public abstract class Command

{

protected SimpleCalculator receiver;

public Command(SimpleCalculator receiver)

{

this.receiver = receiver;

}

public abstract int Execute();

}

public enum CommandOption

{

Add, Substract, Multiply, Divide

}

public class SimpleCalculator

{

private int \_x, \_y;

public SimpleCalculator(int a, int b)

{

\_x = a;

\_y = b;

}

public int Add()

{

return \_x + \_y;

}

public int Subtract()

{

return \_x - \_y;

}

public int Multiply()

{

return \_x \* \_y;

}

public int Divide()

{

return \_x / \_y;

}

}

public class AddCommand : Command

{

private SimpleCalculator \_calculator;

public AddCommand(SimpleCalculator calculator) : base(calculator)

{

\_calculator = calculator;

}

public override int Execute()

{

return \_calculator.Add();

}

}

public class SubtractCommand : Command

{

private SimpleCalculator \_calculator;

public SubtractCommand(SimpleCalculator calculator) :

base(calculator)

{

\_calculator = calculator;

}

public override int Execute()

{

return \_calculator.Subtract();

}

}

public class MultiplyCommand : Command

{

private SimpleCalculator \_calculator;

public MultiplyCommand(SimpleCalculator calculator) :

base(calculator)

{

\_calculator = calculator;

}

public override int Execute()

{

return \_calculator.Multiply();

}

}

public class DivideCommand : Command

{

private SimpleCalculator \_calculator;

public DivideCommand(SimpleCalculator calculator) :

base(calculator)

{

\_calculator = calculator;

}

public override int Execute()

{

return \_calculator.Divide();

}

}

public class Invoker

{

private Command \_command;

public void SetCommand(Command command)

{

\_command = command;

}

public int Execute()

{

return \_command.Execute();

}

}

class Program

{

static void Main(string[] args)

{

SimpleCalculator calculator = new SimpleCalculator(15, 3);

var addCommand = new AddCommand(calculator);

var substractCommand = new SubtractCommand(calculator);

var multiplyCommand = new MultiplyCommand(calculator);

var divideCommand = new DivideCommand(calculator);

Invoker invoker = new Invoker();

invoker.SetCommand(addCommand);

Console.WriteLine("Result is {0}", invoker.Execute());

invoker.SetCommand(substractCommand);

Console.WriteLine("Result is {0}", invoker.Execute());

invoker.SetCommand(multiplyCommand);

Console.WriteLine("Result is {0}", invoker.Execute());

invoker.SetCommand(divideCommand);

Console.WriteLine("Result is {0}", invoker.Execute());

Console.ReadLine();

}

}

}