**public** **abstract** **class** **Command**  
    {  
        **protected** **SimpleCalculator** receiver;  
        **public** **Command**(**SimpleCalculator** receiver)  
        {  
            **this**.receiver = receiver;  
        }  
        **public** **abstract** **int** **Execute**();  
    }

The following enum shows the operations that will be supported in our simple calculator.

**public** **enum** **CommandOption**  
    {  
        **Add**, **Substract**, **Multiply**, **Divide**  
    }

**Create the Receiver class in C#**

The following is a class named SimpleCalculator. This class acts as the Receiver and contains the definition of the Add, Subtract, Multiply, and Divide methods.

**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;  
        }  
    }

**Create the concrete command classes in C#**

The concrete command classes extend the Command abstract base class and implement the Execute method as shown below.

**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**();  
        }  
    }

**Create the Invoker class in C#**

The following code snippet illustrates the Invoker class. It contains two methods, SetCommand and Execute. While SetCommand is used to assign the command object to the private Command reference in the Invoker class, Execute is used to execute the command.

**public** **class** **Invoker**  
    {  
        **private** **Command** \_command;  
        **public** **void** **SetCommand**(**Command** command)  
        {  
            \_command = command;  
        }  
        **public** **int** **Execute**()  
        {  
            **return** \_command.**Execute**();  
        }  
    }

**The command design pattern in action in C#**

Finally, the following code snippet illustrates how you can perform a simple calculation using the SimpleCalculator class.

**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**();  
        }

The command design pattern provides support for extensibility and reduces the coupling that exists between the invoker and receiver of a command. Since the request is encapsulated into a stand-alone object, you can parameterize methods with different requests, save requests in a queue, and even provide support for redo-able or undo-able operations.

—