Strategy
Define a family of algorithms, encapsulating each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

*Not* to be confused with...

Strategery
Define a family of algorithmics, encapsulating each one, and make them interjectionable. Strategery lets the algorithmic vary interdepartmentally from clients that use it.

Design Patterns (Gamma, Helm, Johnson, Vlissides)
In normal words...

If you have an object that can do something in many different ways, rather than putting each way into a method in the object and getting confused, put each way of performing the behavior in its own class.
Generic Strategy Pattern

Configured with a ConcreteStrategy object. Maintains a reference to a Strategy object.

Declares interface common to all supported algorithms. Used by Context.

Implements algorithm using Strategy interface.

Modified from Design Patterns (Gamma, Helm, Johnson, Vlissides)
PowerPoint Strategy Pattern (Example)

Modified from Design Patterns (Gamma, Helm, Johnson, Vlissides)
Instead of this

override
PresentPowerpoint()

override
PresentPowerpoint()

override
PresentPowerpoint()
You have this

PerfectPowerpoint
PresentPerfectPowerpoint()

AlrightPowerpoint
PresentAlrightPowerpoint()

LamePowerpoint
PresentLamePowerpoint()
using System;
using System.Collections.Generic;
using System.Text;

namespace StrategyExample
{
    class Program
    {
        static void Main()
        {
            Presenter p = new Presenter();

            p.SetPresentStrategy(new PerfectPowerPointPresentation());
            p.Present();

            p.SetPresentStrategy(new AlrightPowerPointPresentation());
            p.Present();

            p.SetPresentStrategy(new LamePowerPointPresentation());
            p.Present();

            Console.ReadKey();
        }
    }
}
abstract class PresentStrategy
{
    public abstract void Present();
}

class PerfectPowerPointPresentation : PresentStrategy
{
    public override void Present()
    {
        Console.WriteLine("Perfect: The presentation presented was incredible!!!");
    }
}

class AlrightPowerPointPresentation : PresentStrategy
{
    public override void Present()
    {
        Console.WriteLine("Alright: The presentation presented was just alright");
    }
}

class LamePowerPointPresentation : PresentStrategy
{
    public override void Present()
    {
        Console.WriteLine("Lame: The presentation presented was the one you’re watching now.");
    }
}
class Presenter
{
    private PresentStrategy _presentStrategy;

    public void SetPresentStrategy(PresentStrategy presentStrategy)
    {
        this._presentStrategy = presentStrategy;
    }

    public void Present()
    {
        _presentStrategy.Present();
        Console.WriteLine();
    }

}
Results from Example

Perfect: The presentation presented was incredible!!!
Alright: The presentation presented was just alright.
Lame: The presentation presented was the one you're watching now.
When do I use this?

- When many related classes differ only in their behavior.
- You need different variants of an algorithm.
- An algorithm uses data that clients shouldn’t know about.
- In place of a conditional statement

Modified from Design Patterns (Gamma, Helm, Johnson, Vlissides)