# Simple Factory vs Factory Method vs Abstract Factory in C#

This document compares the three factory-related design patterns in C# — Simple Factory, Factory Method, and Abstract Factory — along with concise examples and key differences.

## 1. Comparison Table

|  |  |  |  |
| --- | --- | --- | --- |
| Aspect | Simple Factory | Factory Method | Abstract Factory |
| Definition | Centralized class creates instances of different concrete classes based on input parameters. | Defines an interface for creating an object but lets subclasses decide which class to instantiate. | Provides an interface for creating families of related or dependent objects without specifying their concrete classes. |
| Design Pattern Type | Not a GoF pattern (basic creational). | GoF Creational Pattern. | GoF Creational Pattern. |
| Class Responsibility | One class (factory) handles object creation for multiple types. | Object creation delegated to subclasses through factory method. | Encapsulates creation of related product families in factory classes. |
| Object Creation Control | Centralized in one factory class. | Decentralized — subclasses decide instantiation. | Decentralized — multiple factories handle related products. |
| Extensibility | Low — modifying the factory class for each new product. | Medium — extend by adding new subclass. | High — can add new product families easily. |
| When to Use | When creation logic is simple and centralized. | When creation logic should vary per subclass. | When creating related objects that must work together. |
| Example Analogy | Kitchen that decides which dish to make based on order type. | Each branch decides how to make its own specialty dish. | Franchise with its own kitchen setup and related dishes. |

## 2. C# Code Examples

### 🔹 Simple Factory

A single factory class creates multiple types of objects based on input.

public interface IShape { void Draw(); }  
  
public class Circle : IShape { public void Draw() => Console.WriteLine("Drawing Circle"); }  
public class Square : IShape { public void Draw() => Console.WriteLine("Drawing Square"); }  
  
public class ShapeFactory {  
 public static IShape CreateShape(string type) {  
 return type switch {  
 "Circle" => new Circle(),  
 "Square" => new Square(),  
 \_ => throw new ArgumentException("Invalid shape type")  
 };  
 }  
}  
  
class Program {  
 static void Main() {  
 IShape shape = ShapeFactory.CreateShape("Circle");  
 shape.Draw();  
 }  
}

### 🔹 Factory Method

Defines an interface for creating an object, but allows subclasses to alter the type of objects created.

public abstract class Page { }  
public class ReportPage : Page { }  
public class ResumePage : Page { }  
  
public abstract class Document {  
 public abstract Page CreatePage();  
}  
  
public class Report : Document {  
 public override Page CreatePage() => new ReportPage();  
}  
  
public class Resume : Document {  
 public override Page CreatePage() => new ResumePage();  
}  
  
class Program {  
 static void Main() {  
 Document doc = new Report();  
 Page page = doc.CreatePage();  
 Console.WriteLine(page.GetType().Name); // Output: ReportPage  
 }  
}

### 🔹 Abstract Factory

Used to create families of related or dependent objects without specifying their concrete classes.

public interface IButton { void Render(); }  
public interface ITextbox { void Render(); }  
  
public class WinButton : IButton { public void Render() => Console.WriteLine("Render Windows Button"); }  
public class WinTextbox : ITextbox { public void Render() => Console.WriteLine("Render Windows Textbox"); }  
  
public class MacButton : IButton { public void Render() => Console.WriteLine("Render Mac Button"); }  
public class MacTextbox : ITextbox { public void Render() => Console.WriteLine("Render Mac Textbox"); }  
  
public interface IUIFactory {  
 IButton CreateButton();  
 ITextbox CreateTextbox();  
}  
  
public class WinFactory : IUIFactory {  
 public IButton CreateButton() => new WinButton();  
 public ITextbox CreateTextbox() => new WinTextbox();  
}  
  
public class MacFactory : IUIFactory {  
 public IButton CreateButton() => new MacButton();  
 public ITextbox CreateTextbox() => new MacTextbox();  
}  
  
class Program {  
 static void Main() {  
 IUIFactory factory = new MacFactory();  
 IButton button = factory.CreateButton();  
 ITextbox textbox = factory.CreateTextbox();  
 button.Render();  
 textbox.Render();  
 }  
}