**Factory Pattern.** A factory creates objects. We implement the factory design pattern in a C# program. With this pattern, we develop an abstraction that isolates the logic for determining which type of class to create.[**Object**](https://www.dotnetperls.com/object)  
**Example.** The factory design pattern relies on a type hierarchy. The classes must all implement an interface or derive from a base class. We use an abstract class as the base. The Manager, Clerk and Programmer classes derive from Position.[**Interface**](https://www.dotnetperls.com/interface)[**Abstract**](https://www.dotnetperls.com/abstract)

**Based on:** .NET 4.5

**Factory pattern example: C#**

**Output**

Where id = 0, position = Manager

Where id = 1, position = Clerk

Where id = 2, position = Clerk

Where id = 3, position = Programmer

**The factory design pattern** is found in the Factory class. The point of the Get method is to take a value and instantiate a class based on that value. It translates integers to objects with a switch statement.[**Switch**](https://www.dotnetperls.com/switch)

**Also:**Because Manager, Clerk, and Programmer all derive from the same abstract class, the return type Position can be used.

**And:**An implicit cast automatically casts the Manager, Clerk and Programmer to Position references.

**The Main method** serves to demonstrate the Factory class in action. We use as part of the demonstration the integers 0, 1, 2, and 3. We use the Get method with each of these values.

**Then:**We show that the appropriate type of class was instantiated for each integer.

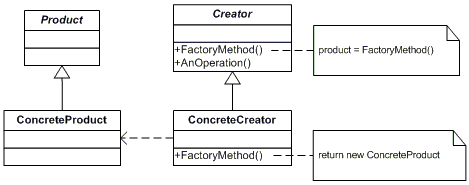
**Discussion.** Imagine you have a system that needs to create objects in many different places. Suppose the system has integers and you want objects for those integers. The factory pattern is ideal for this usage.

**Tip:**You can use the Factory type to handle object creation in a uniform and concise way.

**Summary.** We looked the factory design pattern, which is used to instantiate objects based on another data type such as integers. Factories can be used to reduce code bloat and also make it easier to modify which objects need to be created.

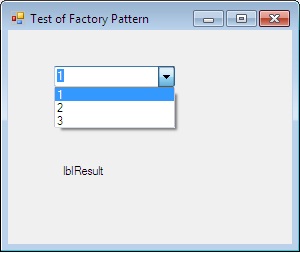
 The classes and objects participating in this pattern are:

An example



Here is the step by step procedure to create an application with Factory Pattern:

1. Create a new Windows project (say: ProFactoryPattern).
2. Add a ComboBox (name it cmbSelect) and one Label (for displaying the Result (name it lblResult)) as shown below:



1. Add the below interface and two classes implement this interface. Please note that both classes have the same methods.

Hide   Copy Code

interface IGet

{

string ConC(string s1, string s2);

}

class clsFirst : IGet

{

public string ConC(string s1, string s2)

{

string Final = "From First: " + s1+" and " + s2;

return Final;

}

}

class clsSecond : IGet

{

public string ConC(string s1, string s2)

{

string Final = "From Second: " + s1 + " and " + s2;

return Final;

}

}

1. Next, add the factory class, in which conditions are there to create the objects. Method in this class basically decides object of class to be created, as shown below:

Hide   Copy Code

class clsFactory

{

static public IGet CreateandReturnObj(int cChoice)

{

IGet ObjSelector = null;

switch (cChoice)

{

case 1:

ObjSelector = new clsFirst();

break;

case 2:

ObjSelector = new clsSecond();

break;

default:

ObjSelector = new clsFirst();

break;

}

return ObjSelector;

}

1. Finally, the client code looks like shown below. In this, the client does not bother about the classes and class name and does not worry if any new class will be added:

Hide   Copy Code

private void cmbSelect\_SelectedIndexChanged(object sender, EventArgs e)

{

IGet ObjIntrface = null;

ObjIntrface = clsFactory.CreateandReturnObj(cmbSelect.SelectedIndex + 1);

string res = ObjIntrface.ConC("First", "Second");

lblResult.Text = res;

}

1. Now the output will be:

If 1 will be selected, then object of clsFirst will be created and in case of 2, clsSecond will be created:

Design patterns are general reusable solutions to common problems that occurred in software designing. There are broadly 3 categories of design patterns, i.e., Creational, Behavioral and Structural.

Now, Factory Design Pattern falls under the category of creational design pattern.  
It deals with the problem of creating objects (products) without specifying the exact class of object that will be created. The essence of this pattern is to "Define an interface for creating an object, but let the classes that implement the interface decide which class to instantiate. The Factory method lets a class defer instantiation to sub classes."

## Where to Use?

It would be tedious when the client needs to specify the class name while creating the objects. So, to resolve this problem, we can use Factory pattern. It provides the client a simple way to create the object. The example below will elaborate the factory pattern in detail.