Abstract Factory Pattern

By Md. Mahedee Hasan, Software Architect, Leadsoft Bangladesh Ltd.
Source: http://mahedee.net/abstract-factory-pattern/

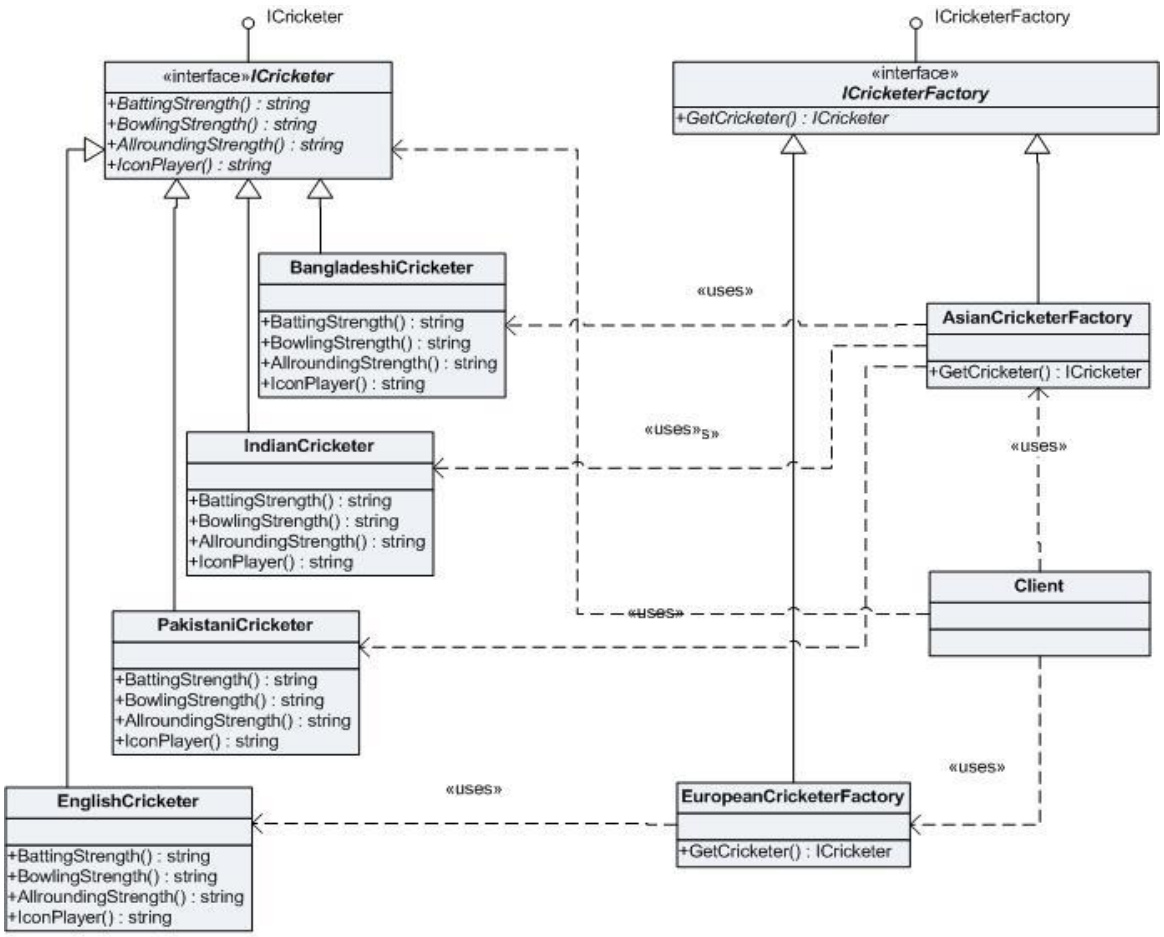
Abstract factory pattern is a creational design pattern. Creational design pattern is deals with object creation mechanism. Object creation mechanism is changed on the basis of problem. Abstract factory pattern provides an interface to create families of related or dependent objects without specifying their concrete class. It is identified on the basis of problem complexity. It is encapsulated the process of instantiation of family of classes. Abstract factory pattern is widely used in framework and libraries to encapsulate the family of classes.

Elements of Abstract Factory Pattern:

1. Abstract Factory – An Interface to create families of related or dependent item object.
2. Concrete Factory – Implement the interface to create families of related item object.
3. Abstract Item – An interface to create concrete item object.
4. Concrete Item – Implement the interface to create concrete item object.
5. Client – Uses Interface (abstract Item) provided by abstract factory and access concrete object by this interface.

Implementation:

Here I implemented abstract factory pattern in International Cricket Team by C#. The UML diagram is given below.



Abstract Factory Pattern
UML Class Diagram

Step 1: Create interface for Item class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AFP
{
    /// <summary>
    /// Interface for Item class
    /// </summary>

    public interface ICricketer
    {
        string BattingStrength();
        string BowlingStrength();
        string AllroundingStrength();
        string IconPlayer();
    }
```

```
}
```

Step 2: Create Factory Interface

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AFP
{
    /// <summary>
    /// A Factory interface
    /// </summary>
    public interface ICricketerFactory
    {
        ICricketer GetCricketer(CricketerBase cricketerBase);
    }
}
```

Step 3: Define the type of Base object

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AFP
{

    /// <summary>
    /// Define the type of Base Object
    /// </summary>
    public enum CricketerBase
    {
        AustralianCricketer,
        BangladeshiCricketer,
        EnglishCricketer,
        IndianCricketer,
        PakistaniCricketer
    }

}
```

Step 4: Create `Concrete factory class for Asian Cricketer`

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace AFP
{

    /// <summary>
    /// Concrete factory class for Asian Cricketer
    /// </summary>

    public class AsianCricketerFactory : ICricketerFactory
    {

        #region ICricketerFactory Members

        public ICricketer GetCricketer(CricketerBase cricketerBase)
        {
            ICricketer objICricketer = null;

            switch (cricketerBase)
            {
                case CricketerBase.BangladeshiCricketer:
                    objICricketer = new BangladeshiCricketer();
                    break;
                case CricketerBase.IndianCricketer:
                    objICricketer = new IndianCricketer();
                    break;
                default:
                    break;
            }
            return objICricketer;
        }

        #endregion
    }
}
```

Step 5: Create Concrete factory class for European Cricketer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AFP
{
    /// <summary>
    /// Concrete factory class for European Cricketer
    /// </summary>

    public class EuropeanCricketerFactory : ICricketerFactory
    {

        #region ICricketerFactory Members
```

```
        public ICricketer GetCricketer(CricketerBase cricketerBase)
        {
            ICricketer objICricketer = null;

            switch (cricketerBase)
            {
                case CricketerBase.EnglishCricketer:
                    objICricketer = new EnglishCricketer();
                    break;
                default:
                    break;
            }
            return objICricketer;
        }

        #endregion
    }
}
```

Step 6: Create Item class for Bangladeshi Cricketer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AFP
{

    /// <summary>
    /// Item class for Bangladeshi Cricketer
    /// </summary>

    public class BangladeshiCricketer : ICricketer
    {
        #region ICricketer Members

        public string BattingStrength()
        {
            return "60%";
        }

        public string BowlingStrength()
        {
            return "70%";
        }

        public string AllroundingStrength()
        {
            return "85%";
        }

        public string IconPlayer()
        {
```

```
            return "Shakib Al Hasan";
        }

        #endregion
    }
}
```

Step 7: Create Item Class for Indian Cricketer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AFP
{

    /// <summary>
    /// Item class for Indian Cricketer
    /// </summary>

    public class IndianCricketer : ICricketer
    {

        #region ICricketer Members

        public string BattingStrength()
        {
            return "85%";
        }

        public string BowlingStrength()
        {
            return "60%";
        }

        public string AllroundingStrength()
        {
            return "70%";
        }

        public string IconPlayer()
        {
            return "Shachin Tendulkar.";
        }

        #endregion
    }
}
```

Step 8: Create Item class for Pakistani Cricketer

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace AFP
{

    /// <summary>
    /// Item class for Pakistani Cricketer
    /// </summary>

    public class PakistaniCricketer : ICricketer
    {

        #region ICricketer Members

        public string BattingStrength()
        {
            return "75%";
        }

        public string BowlingStrength()
        {
            return "85%";
        }

        public string AllroundingStrength()
        {
            return "75%";
        }

        public string IconPlayer()
        {
            return "Shahid Afridi.";
        }

        #endregion
    }
}
```

Step 9: Create Item class for English Cricketer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AFP
{

    /// <summary>
    /// Item class for English Cricketer
    /// </summary>

    public class EnglishCricketer : ICricketer
    {
```

```
        #region ICricketer Members

        public string BattingStrength()
        {
            return "75%";
        }

        public string BowlingStrength()
        {
            return "80%";
        }

        public string AllroundingStrength()
        {
            return "70%";
        }

        public string IconPlayer()
        {
            return "Kavin Pietersen";
        }

        #endregion
    }
}
```

Step 10: Create a client class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AFP
{

    /// <summary>
    /// Client Class
    /// </summary>

    class Program
    {
        static void Main(string[] args)
        {
            AsianCricketerFactory objAsianFactory = new AsianCricketerFactory();
            ICricketer objIAsianCricketer =
objAsianFactory.GetCricketer(CricketerBase.BangladeshiCricketer);
            Console.WriteLine("Bangladesh Cricket Team\nBatting Strength:" +
objIAsianCricketer.BattingStrength());
            Console.WriteLine("Bowling Strength:" +
objIAsianCricketer.BowlingStrength());
            Console.WriteLine("Allrounding Strength:" +
objIAsianCricketer.AllroundingStrength());
            Console.WriteLine("Icon Player:" + objIAsianCricketer.IconPlayer());
```

```
            Console.WriteLine();

            EuropeanCricketerFactory objEuropeanFactory = new EuropeanCricketerFactory();
            ICricketer objIEuropeanCricketer =
objEuropeanFactory.GetCricketer(CricketerBase.EnglishCricketer);
            Console.WriteLine("England Cricket Team\nBatting Strength:" +
objIEuropeanCricketer.BattingStrength());

            Console.WriteLine("Bowling Strength:" +
objIEuropeanCricketer.BowlingStrength());
            Console.WriteLine("Allrounding Strength:" +
objIEuropeanCricketer.AllroundingStrength());
            Console.WriteLine("Icon Player:" + objIEuropeanCricketer.IconPlayer());


            Console.ReadLine();


        }
    }
}
```

Step 11: Output