**Tên, phân loại:**

* Tên: Builder Pattern
* Phân loại: Creational Pattern.

**Mục đích, ý định:**

* “Tách rời việc tạo dựng một đối tượng phức tạp ra khỏi bản thân đối tượng, vì vậy cho phép cùng một quá trình tạo dựng có thể tạo ra nhiều loại đối tượng khác nhau”

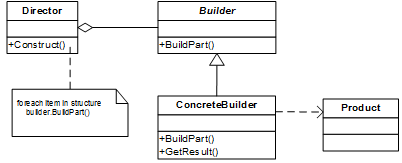
**Khi nào sử dụng:**

* Thuật toán cho việc tạo ra một đối tượng phức tạp bị phụ thuộc vào nhiều thành phần độc lập tạo nên đối tượng và làm thế nào các thành phần đó kết hợp lại với nhau.
* Quá trình xây dựng nên một đối tượng cần phải cho phép nhiều cách biểu diễn (representation) khác nhau.
* Khi client mong muốn sẽ kiểm soát được quá trình tạo dựng đối tượng.

**Khả năng ứng dụng:**

* Ứng dụng trong các hệ thống lớn, nhiều chức năng phức tạp và mô hình giao diện đồ sộ và việc khởi tạo ứng dụng thường rất khó khăn.
* Quá trình khởi tạo một đối tượng cần trải qua nhiều công đoạn khác nhau, và chúng ta có nhu cầu kiểm soát các biến thể khác nhau của quá trình này.

**Cấu trúc:**

****

**Các thành viên:**

* **Builder**: định nghĩa một lớp trừu tượng cho việc khởi tạo các thành phần của một đối tượng sản phẩm Product sẽ được nó tạo ra.
* **ConcreteBuilder**: là lớp con kế thừa từ lớp Builder để override lại hàm tạo từng thành phần của một sản phẩm, lắp ráp các phần vào sản phẩm, nó sẽ chứa một tham chiếu đến đối tượng sản phẩm bên trong nó, đồng thời đưa ra một phương thức để thể hiện nó ra.
* **Product**: là sản phẩm bao gồm nhiều phần cấu hình nên.
* **Director**: là nơi gọi xây dựng các phần để tạo ra một Product thông qua tham chiếu đến đối tượng Builder.

**Sự cộng tác:**

* Client tạo ra các đối tượng Director và cấu hình nó với các đối tượng Builder mong muốn.
* Builder nhận các yêu cầu từ director gọi đến nó và tiến hành xây dựng từng phần của sản phẩm.
* Client nhận kết quả được trả về từ builder.

**Các hệ quả mang lại:**

* Builder để cho chúng ta làm thay đổi cách thức thể hiện bên trong một sản phẩm mà nó xây dựng cho chúng ta:
* Mỗi Builder độc lập với nhau và phần còn lại của chương trình, điều này làm tăng tính mô đun hóa và việc thêm các Builder khác có liên quan sẽ dễ dàng hơn rất nhiều.
* Mỗi Builder tạo nên một sản phẩm cuối cùng theo từng bước và phụ thuộc vào dữ liệu, chúng ta có toàn quyền điều khiển mỗi sản phẩm được tạo ra bởi Builder.

**Chú ý liên quan đến việc cài đặt:**

* Cần chú ý sự khác biệt nhau giữa mẫu Builder và mẫu Abstract Factory: mẫu Builder được client chỉ thị làm cách nào để tạo ra đối tượng (bằng cách chỉ rõ các lớp nào sẽ được dùng trong khi quá trình tạo) trong khi Abstract Factory được client sử dụng để tạo ra đối tượng trực tiếp.
* Trong thực tế, các sản phẩm được tạo ra bởi các builder thường có các cấu trúc bên trong khác biệt nhau rất lơn. Do đó các sản phẩm này không cần thiết phải được dẫn xuất từ một lớp cha chung.

**Ví dụ về các hệ thống thực tế:**

**Các mẫu liên quan:**

* Một mẫu Builder hơi giống với một mẫu Abstract Factory khi cả hai đều trả về một đối tượng thuộc class được trang điểm bằng một số các phương thức và các đối tượng khác.
* Sự khác biệt chính giữa chúng là trong khi Abstract  Factory trả về một đối tượng thuộc cùng một cây phân cấp thì Builder trả về một đối tượng phức tạp được tạo từng bước từng bước, phụ thuộc vào dữ liệu thể hiện đối tượng đó.

**Mã nguồn minh họa:**

**Bài toán:**

Ở trung tâm chăm sóc của Elizabeth’s, thầy cô giáo sẽ giúp trẻ em làm toàn bộ các loại đồ chơi để phát triển khả năng sáng tạo của chúng. Một hoạt động ưa thích của Alizabeth đó là tạo ra các loại động vật bằng đất sét đồ chơi.

Các bộ khuôn là dụng cụ mà Elizabeth luôn dùng để tạo ra các loại động vật ưa thích của cô.

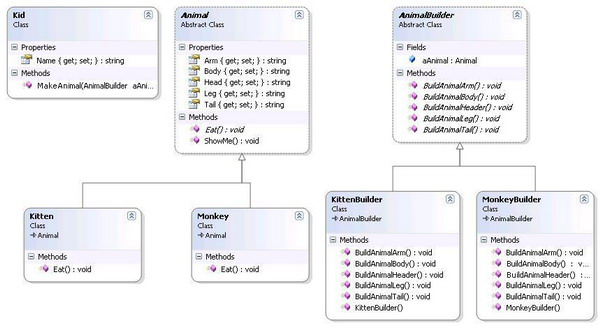
Một bộ khuôn đầy đủ thì bao gồm 5 phần đầu, thân, cánh tay, chân và đuôi. Khi Eli muốn làm một con vật thì cô sẽ dùng mỗi khuôn để tạo ra đầu, thân, chân, tay và đuôi cho đông vật đó. Có nhiều bộ khuôn động vật khác nhau mà trẻ em có thể chọn.

Ví dụ: Nếu Eli muốn tạo ra một con khỉ, thì cô sẽ chọn bộ khuôn của khỉ để bắt đầu:

* Bước 1. Tạo ra đầu con khỉ.
* Bước 2. Tạo ra thân con khỉ.
* Bước 3. Tạo ra chân con khỉ.
* Bước 4. Tạo ra tay con khỉ.
* Bước 5. Tạo ra đuôi con khỉ.

Một khi toàn bộ năm phần được hoàn thành, Eli sẽ dán chúng lại và trang trí cho con khỉ đến khi hoàn thành sản phẩm. Cô sẽ tặng nó cho trẻ em một khi con khỉ đã trang trí hoàn toàn. Nếu Eli muốn tạo một con vật khác như Mèo con chẳng hạn, chỉ cần lặp lại các bước trên với bộ khuôn mèo con.

**Sơ đồ lớp và mối quan hệ:**

****

**Code mẫu:**

#### AnimalBuilder

**using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace www.askbargains.com  
{  
   namespace BuilderDesignPattern  
   {  
       public abstract class AnimalBuilder  
       {  
           public Animal aAnimal;  
  
           public abstract void BuildAnimalHeader();  
           public abstract void BuildAnimalBody();  
           public abstract void BuildAnimalLeg();  
           public abstract void BuildAnimalArm();  
           public abstract void BuildAnimalTail();  
       }  
   }  
}**

#### MonkeyBuilder

**using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace www.askbargains.com  
{  
   namespace BuilderDesignPattern  
   {  
       public class MonkeyBuilder : AnimalBuilder  
       {  
  
           public MonkeyBuilder()  
           {  
               aAnimal = new Monkey();  
           }  
  
           public override void BuildAnimalHeader()  
           {  
               aAnimal.Head = "Moneky's Head has been built";  
           }  
  
           public override void BuildAnimalBody()  
           {  
               aAnimal.Body = "Moneky's Body has been built";  
           }  
  
           public override void BuildAnimalLeg()  
           {  
               aAnimal.Leg = "Moneky's Leg has been built";  
           }  
  
           public override void BuildAnimalArm()  
           {  
               aAnimal.Arm = "Moneky's Arm has been built";  
           }  
  
           public override void BuildAnimalTail()  
           {  
               aAnimal.Tail = "Moneky's Tail has been built";  
           }  
       }  
   }  
}**

#### KittenBuilder

**using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace www.askbargains.com  
{  
   namespace BuilderDesignPattern  
   {  
       public class KittenBuilder : AnimalBuilder  
       {  
           public KittenBuilder()  
           {  
               aAnimal = new Kitten();  
           }  
  
           public override void BuildAnimalHeader()  
           {  
               aAnimal.Head = "Kitten's Head has been built";  
           }  
  
           public override void BuildAnimalBody()  
           {  
               aAnimal.Body = "Kitten's Body has been built";  
           }  
  
           public override void BuildAnimalLeg()  
           {  
               aAnimal.Leg = "Kitten's Leg has been built";  
           }  
  
           public override void BuildAnimalArm()  
           {  
               aAnimal.Arm = "Kitten's Arm has been built";  
           }  
  
           public override void BuildAnimalTail()  
           {  
               aAnimal.Tail = "Kitten's Tail has been built";  
           }  
       }  
   }  
}**

#### Animal

**using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace www.askbargains.com  
{  
   namespace BuilderDesignPattern  
   {  
       public abstract class Animal  
       {  
           public string Head { get; set; }  
           public string Body { get; set; }  
           public string Leg { get; set; }  
           public string Arm { get; set; }  
           public string Tail { get; set; }  
  
          
           *//helper method for demo the Polymorphism, so we can*           *//easily tell what type object it is from client.*           public abstract void Eat();  
             
           *//helper method for demo the result from client*           public void ShowMe()  
           {  
               Console.WriteLine(Head);  
               Console.WriteLine(Body);  
               Console.WriteLine(Leg);  
               Console.WriteLine(Arm);  
               Console.WriteLine(Tail);  
               Eat();  
  
           }  
       }  
   }  
}**

#### Monkey Class

**using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace www.askbargains.com  
{  
   namespace BuilderDesignPattern  
   {  
       public class Monkey : Animal  
       {  
           *//helper method to show monkey's property for demo purpose*           public override void Eat()  
           {  
               Console.WriteLine("Since I am Monkey, I like to eat banana");  
           }  
       }  
   }  
}**

#### Kitten Class

**using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace www.askbargains.com  
{  
   namespace BuilderDesignPattern  
   {  
       public class Kitten : Animal  
       {  
           public override void Eat()  
           {  
               Console.WriteLine("Since I am Kitten, I like to eat kitten food");  
           }  
       }  
   }  
}**

### Constructor Object

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace www.askbargains.com  
{  
   namespace BuilderDesignPattern  
   {  
       public class Kid  
       {  
           public string Name { get; set; }  
  
           *//construct process to build an animal object,*           *//after this process completed, a object*           *//will be consider as a ready to use object.*           public void MakeAnimal(AnimalBuilder aAnimalBuilder)  
           {  
               aAnimalBuilder.BuildAnimalHeader();  
               aAnimalBuilder.BuildAnimalBody();  
               aAnimalBuilder.BuildAnimalLeg();  
               aAnimalBuilder.BuildAnimalArm();  
               aAnimalBuilder.BuildAnimalTail();  
           }  
  
  
       }  
   }  
}

### Client App

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using www.askbargains.com.BuilderDesignPattern;  
  
namespace www.askbargains.com  
{  
   namespace Client  
   {  
       class Program  
       {  
           static void Main(string[] args)  
           {  
               *//create a constructor object to start building*               Kid aKid = new Kid();  
               aKid.Name = "Elizabeth";  
  
               *//Elizabeth use Monkey mold to make a monkey*               Console.WriteLine("{0} start making a monkey",aKid.Name);  
               AnimalBuilder builderA = new MonkeyBuilder();  
               aKid.MakeAnimal(builderA);  
               builderA.aAnimal.ShowMe();  
  
               *//Elizabeth use Kitten mold to make a kitten*                Console.WriteLine("{0} start making a kitten",aKid.Name);  
                AnimalBuilder builderB = new KittenBuilder();  
                aKid.MakeAnimal(builderB);  
                builderB.aAnimal.ShowMe();  
  
               Console.Read();  
           }  
       }  
   }  
}

**Output**

