عنوان: **آشنایی با الگوی طراحی Prototype**
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۴ ۱۹:۲۰
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

فرض کنید در حال پختن یک کیک هستید. ابتدا کیک را می‌پذید و سپس آن را تزیین می‌کنید. عملیات پختن کیک، فرآیند ثابتی است و تزیین کردن آن متفاوت. گاهی کیک را با کاکائو تزیین می‌کنید و گاهی با میوه و غیره.
پیش از اینکه سناریو را بیش از این جلو ببریم، وارد بحث کد می‌شویم. طبق سناریوی فوق، فرض کنید کلاسی بنام Prototype دارید که این کلاس هم از کلاس انتزاعی APrototype ارث برده است. در ادامه یک شیء از این کلاس می‌سازید و مقادیر مختلف آن را تنظیم کرده و کار را ادامه می‌دهید.

```
public abstract class APrototype : ICloneable     {
        public string Name { get; set; }
        public string Health { get; set; }
    }

    public class Prototype : APrototype
    {
        public override string ToString() { return string.Format("Player name: {0}, Health statuse:
{1}", Name, Health); }
    }
```

در ادامه از این کلاس نمونه‌گیری می‌کنیم:

```
Prototype p1 = new Prototype { Name = "Vahid", Health = "OK" };
Console.WriteLine(p1.ToString());
```

حالا فرض کنید به یک آبجکت دیگر نیاز دارید، ولی این آبجکت عینا مشابه p1 است؛ لذا نمونه‌گیری، از ابتدا کار مناسبی نیست.
برای اینکار کافیست کدها را بصورت زیر تغییر دهیم:

```
public abstract class APrototype : ICloneable
    {
        public string Name { get; set; }
        public string Health { get; set; }
        public abstract object Clone();
    }

    public class Prototype : APrototype
    {
        public override object Clone() { return this.MemberwiseClone() as APrototype; }
        public override string ToString() { return string.Format("Player name: {0}, Health statuse:
{1}", Name, Health); }
    }
```

در متد Clone از MemberwiseClone استفاده کرده‌ایم. خود Clone هم در داخل واسط ICloneable تعریف شده‌است و هدف از آن کپی نمودن آبجکت‌ها است. سپس کد فوق را بصورت زیر مورد استفاده قرار می‌دهیم:

```
Prototype p1 = new Prototype { Name = "Vahid", Health = "OK" };
Prototype p2 = p1.Clone() as Prototype;
Console.WriteLine(p1.ToString());
Console.WriteLine(p2.ToString());
```

با اجرای کد فوق مشاهده میشود p1 و p2 دقیقا عین هم کار می‌کنند. کل این فرآیند بیانگر الگوی Prototype می‌باشد. ولی تا اینجای کار درست است که الگو پیاده سازی شده است، ولی همچنین به نظر نقصی نیز در کد دیده می‌شود:
برای واضح نمودن نقص، یک کلاس بنام AdditionalDetails تعریف می‌کنیم. در واقع کد را بصورت زیر تغییر میدهیم:

```
public abstract class APrototype : ICloneable
    {
        public string Name { get; set; }
        public string Health { get; set; }
```

```
        public AdditionalDetails Detail { get; set; }
        public abstract object Clone();
    }
    public class AdditionalDetails { public string Height { get; set; } }

    public class Prototype : APrototype
    {
        public override object Clone() { return this.MemberwiseClone() as APrototype; }
        public override string ToString() { return string.Format("Player name: {0}, Health statuse:
{1}, Height: {2}", Name, Health, Detail.Height); }
    }
```

و از آن بصورت زیر استفاده می‌کنیم:

```
Prototype p1 = new Prototype { Name = "Vahid", Health = "OK", Detail = new AdditionalDetails { Height =
"100" } };
Prototype p2 = p1.Clone() as Prototype;
p2.Detail.Height = "200";
Console.WriteLine(p1.ToString());
Console.WriteLine(p2.ToString());
```

خروجی که نمایش داده می‌شود در بخش Height هم برای p1 و هم برای p2 عدد 200 را نمایش می‌دهد که می‌تواند اشتباه باشد. چراکه p1 دارای Height برابر با 100 است و p2 دارای Height برابر با 200. به این اتفاق ShallowCopy گفته میشود که ناشی از استفاده از MemberwiseClone است که در مورد ارجاعات با آدرس رخ می‌دهد. در این حالت بجای کپی نمودن مقدار، از کپی نمودن آدرس استفاده میشود (Ref Type چیست؟)

برای حل این مشکل باید DeepCopy انجام داد. لذا کد را بصورت زیر تغییر می‌دهیم:(DeepCopy و ShallowCopy چیست؟)

```
public abstract class APrototype : ICloneable
    {
        public string Name { get; set; }
        public string Health { get; set; }
        //This is a ref type
        public AdditionalDetails Detail { get; set; }
        public abstract APrototype ShallowClone();
        public abstract object Clone();
    }

    public class AdditionalDetails { public string Height { get; set; } }

    public class Prototype : APrototype
    {
        public override object Clone()
        {
            Prototype cloned = MemberwiseClone() as Prototype;
            //We need to deep copy each ref types in order to prevent shallow copy
            cloned.Detail = new AdditionalDetails { Height = this.Detail.Height };
            return cloned;
        }
        //Shallow copy will copy ref type's address instead of their value, so any changes in cloned
object or source object will take effect on both objects
        public override APrototype ShallowClone() { return this.MemberwiseClone() as APrototype; }
        public override string ToString() { return string.Format("Player name: {0}, Health statuse:
{1}, Height: {2}", Name, Health, Detail.Height); }
    }
```

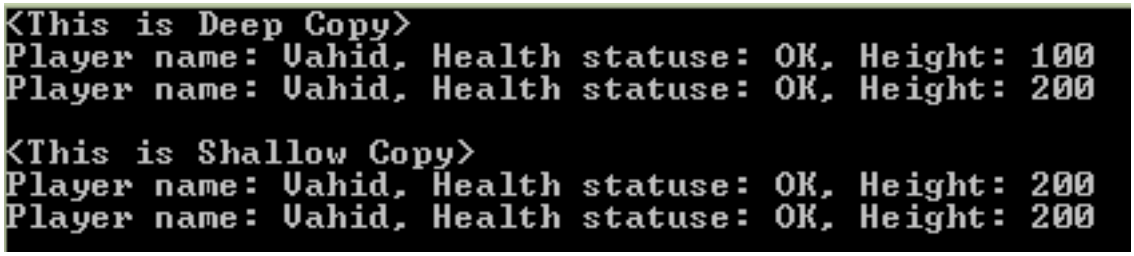
و سپس بصورت زیر از آن استفاده نمود:

```
Prototype p1 = new Prototype { Name = "Vahid", Health = "OK", Detail = new AdditionalDetails { Height =
"100" } };
            Prototype p2 = p1.Clone() as Prototype;
            p2.Detail.Height = "200";
            Console.WriteLine("<This is Deep Copy>");
            Console.WriteLine(p1.ToString());
            Console.WriteLine(p2.ToString());

            Prototype p3 = new Prototype { Name = "Vahid", Health = "OK", Detail = new
AdditionalDetails { Height = "100" } };
            Prototype p4 = p3.ShallowClone() as Prototype;
            p4.Detail.Height = "200";
            Console.WriteLine("\n<This is Shallow Copy>");
```

```
            Console.WriteLine(p3.ToString());
            Console.WriteLine(p4.ToString());
```

لذا خروجی بصورت زیر را می‌توان مشاهده نمود:



البته در این سناریو ShallowCopy باعث اشتباه شدن نتایج می‌شود. شاید شما در دامنه‌ی نیازمندیهای خود، اتفاقا به ShallowCopy نیاز داشته باشید و DeepCopy مرتفع کننده‌ی نیاز شما نباشد. لذا کاربرد هر کدام از آنها وابستگی مستقیمی به دامنه‌ی نیازمندی‌های شما دارد.