Core Principle: Programming to an Interface instead of an Implementation

One of the core principles of object-oriented programming is to always program to an interface instead of a concrete object. Let us understand this in detail. Assume that we have multiple kinds of products in our OMS (Order Management System) such as foods, electronics and beauty products. To encapsulate these different products, we create different classes, such as `FoodProducts`, `ElectronicProducts`, `BeautyProducts`, and so on.

Because each product will have common properties and methods shared by all kinds of products, for example Unit Cost, Size, Weight, and so on, we create an interface called `IProducts` to list the basic behavior of a product.

```
public interface IProduct
{
       float UnitCost { get; set;}
       float Weight   {get; set;}

       bool Update();
}
```

In this interface-`IProduct`- we have defined two basic fields with both getter and setter properties, along with a method. (There can be more, but for the purpose of our understanding, we will work with only these two properties and one method.)

Next, we create individual concrete `product` classes (one for each different product). Here is one such class, `BeautyProduct`:

```
public class BeautyProduct:IProduct
{
   private float _unitCost;
   private float _weight;
   private int _forGender;
   public float UnitCost
   {
     get { return _unitCost; }
     set { _unitcost = value; }
   }

  public float Weight
   {
     get { return _ weight; }
     set { _ weight = value; }
   }
```

```
public int ForGender
 {
   get { return _ forGender; }
   set { _ forGender = value; }
 }

public bool Update()
{
 try
   {
    //code to update the product calling DAL method
    return ProductDAL.Update(this);
   }
  catch(Exception ex)
  {
      //log and rethrow…
  }
 }
```

In the above concrete class, we have implemented the `IProduct` interface with concrete methods and properties. Note that we have added a property called `ForGender`, which tells us the gender group to which this beauty product belongs. This can be male, female or unisex, with each value being represented by an integer, as in 1, 2, or 3.

Because this is a beauty product specific property, it is not present in the interface, `IProduct`.

Now if we want to use `BeautyProduct` objects in our code, the first and simple approach would be:

```
BeautyProduct bp = new BeautyProduct();
```

Let us understand this line in more detail. The `new` keyword is used to create a new instance of a class. When using the `new` keyword, there are two important points to consider—the "type" of the object on the left-side and the right-side of the `new` keyword. In the above case, we are programming to a concrete implementation, that is, `BeautyProduct`, so the type of the object `bp` is `BeautyProduct`, and the implementation type is also `BeautyProduct` (as we have used in this syntax, `new BeautyProduct()`).