OO Principles (2)

ELIZABETH BOURKE

OOP1.Product

```
6
7   namespace OOP1
8   {
        10 references
9       class Product
10
11      {
            3 references
12          public string Code { get; set; }
            2 references
13          public string Description { get; set; }
            2 references
14          public decimal Price { get; set; }
15
16
17
            0 references
18          public Product() { }
19
            4 references
20          public Product(string c, string d, decimal p)
21          {
22              this.Code = c;
23              this.Description = d;
24              this.Price = p;
25          }
26
            4 references
27          public void Print()
28          {
29              Console.WriteLine("Code" + this.Code);
30              Console.WriteLine("Description" + this.Description);
31              Console.WriteLine("Price" + this.Price);
32          }
33      }
34  }
35
```

Data members - should be created private

Data Encapsulation

Here the public is used to create public methods that will allow access to the private datamember

Auto-Implemented Properties

In C# 3.0 and later, auto-implemented properties make property-declaration more concise when no additional logic is required in the property accessors. They also enable client code to create objects. When you declare a property as shown in the following example, the compiler creates a private, anonymous backing field that can only be accessed through the property's `get` and `set` accessors.

```
namespace OOP2
{
    // This class is mutable. Its data can be modified from
    // outside the class.
    3 references
    class Customer
    {
        // Auto-Impl Properties for trivial get and set
        2 references
        public double TotalPurchases { get; set; }
        1 reference
        public string Name { get; set; }
        1 reference
        public int CustomerID { get; set; }

        // Constructor
        1 reference
        public Customer(double purchases, string name, int ID)
        {
            TotalPurchases = purchases;
            Name = name;
            CustomerID = ID;
        }
        // Methods
        0 references
        public string GetContactInfo() { return "ContactInfo"; }
        0 references
        public string GetTransactionHistory() { return "History"; }

        // .. Additional methods, events, etc.
    }

    0 references
    class Program
    {
        0 references
        static void Main()
        {
            // Intialize a new object.
            Customer cust1 = new Customer(4987.63, "Northwind", 90108);

            //Modify a property
            cust1.TotalPurchases += 499.99;
        }
    }
}
```

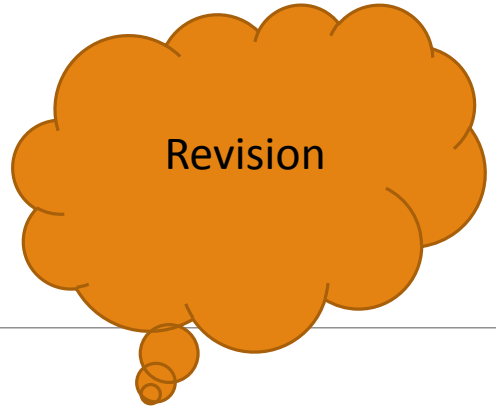
The class that is shown in the this example is mutable. Client code can change the values in objects after they are created. For small classes that just encapsulate a set of values (data) and have little or no behaviors, you should either make the objects immutable by declaring the set accessor as private (immutable to consumers) or by declaring only a get accessor (immutable everywhere except the constructor).

```
// This class is immutable. After an object is created,
// it cannot be modified from outside the class. It uses a
// constructor to initialize its properties.
2 references
class Contact
{
    // Read-only properties.
    2 references
    public string Name { get; }
    2 references
    public string Address { get; private set; }

    // Public constructor.
    1 reference
    public Contact(string contactName, string contactAddress)
    {
        Name = contactName;
        Address = contactAddress;
    }
}
```

```
private string name;
public string Name
{
    get
    {
        return this.name;
    }
    set
    {
        this.name = value;
    }
}
```

Arrays in C Sharp



The syntax for creating a one-dimensional array

With two statements

```
type[] arrayName;                    // declaration statement
arrayName = new type[arrayLength];   // assignment statement
```

With one statement

```
type[] arrayName = new type[arrayLength];
```

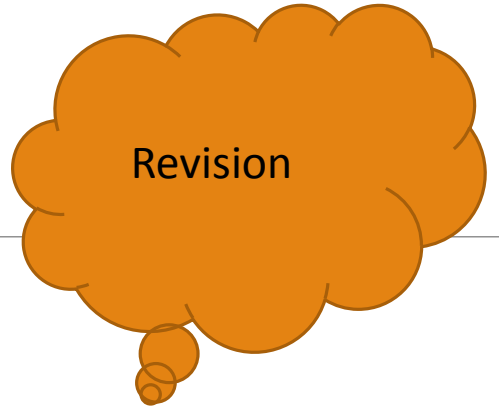
Examples that create an array of decimal types

With two statements

```
decimal[] totals;
totals = new decimal[4];
```

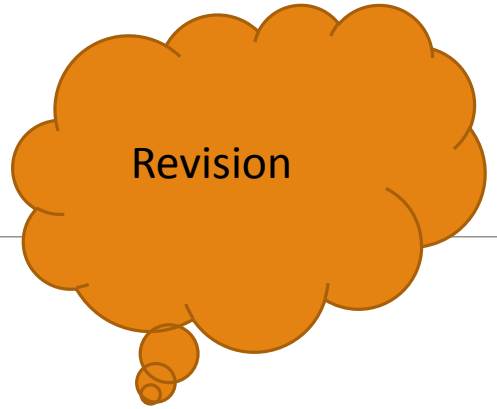
With one statement

```
decimal[] totals = new decimal[4];
```



Code that puts the numbers 0 through 9 into an array

```
int[] numbers = new int[10];
for (int i = 0; i < numbers.Length; i++)
    numbers[i] = i;
```



The syntax of a foreach loop

```
foreach (type elementName in arrayName)
{
    statements
}
```

Code that computes the average of the totals array

```
decimal sum = 0.0m;
foreach (decimal total in totals)
    sum += total;
decimal average = sum/totals.Length;
```

Array of Objects

```
// create an array of objects
Product[] cat1 = new Product[2];

for (int i = 0; i < cat1.Length; i++)
{
    Product product = new Product();
    product.getUserInput();
    cat1[i] = product;
}

foreach (Product product in cat1)
{
    product.Print();
}
```

```
class Product

{
    3 references
    public string Code { get; set; }
    3 references
    public string Description { get; set; }
    3 references
    public decimal Price { get; set; }



    1 reference
    public Product() { }

    0 references
    public Product(string c, string d, decimal p)
    {
        this.Code = c;
        this.Description = d;
        this.Price = p;
    }
    1 reference
    public void getUserInput()
    {
        Console.WriteLine("Please enter product code");
        string pc = Console.ReadLine();
        Console.WriteLine("Please enter product description");
        string pd = Console.ReadLine();
        Console.WriteLine("Please enter product price");
        string pp = Console.ReadLine();

        this.Code = pc;
        this.Description = pd;
        this.Price = Convert.ToDecimal(pp);

    }
    1 reference
    public void Print()
    {
        Console.WriteLine("Code" + this.Code);
        Console.WriteLine("Description" + this.Description);
        Console.WriteLine("Price" + this.Price);
    }
}
```

Search by code ..

Allow the user to enter a product code the system should display the corresponding product details for the product/

```
Product[] cat1 = new Product[4];
//test data
Product p = new Product("DF123", "Power Cables yellow", 12.99m);
Product p1 = new Product("DF124", "Power Cables green", 13.99m);
Product p2 = new Product("DF125", "Power Cables red", 14.99m);
Product p3 = new Product("DF126", "Power Cables blue", 15.99m);
cat1[0] = p;
cat1[1] = p1;
cat1[2] = p2;
cat1[3] = p3;

Console.WriteLine("enter product code");
string pc = Console.ReadLine();
//find product code in array
foreach (Product product in cat1)
{
    if (product.Code.Equals(pc))
    {
        Console.WriteLine("Product found");
        product.Print();
    }
}
```

Update..

Search and update

```
//find product code in array
foreach (Product product in cat1)
{
    if (product.Code.Equals(pc))
    {
        Console.WriteLine("Product found");
        product.getUserInput();
    }
}
```

Bookshop Lab