c# class layout

```
using System;
using System.ComponentModel;

namespace Examples
{
    public sealed class Customer : INotifyPropertyChanged
    {
        public Customer() { }

        public Customer(String firstName,
                        String lastName,
                        DateTime dateOfBirth)
        {
            FirstName = firstName;
            LastName = lastName;
            DateOfBirth = dateOfBirth;
        }

        public String FirstName
        {
            get { return this.firstName; }
            set { this.firstName = value; }
        }

        public String LastName { get;  set; }

        internal DateTime? DateOfBirth { get; set; }

        public event PropertyChangedEventHandler PropertyChanged;

        public override string ToString()
        {
            return FirstName + " " + LastName;
        }

        public Int32 GetAge()
        {
            if (DateOfBirth == null)
            {
                throw new NullReferenceException(
                                "DateOfBirth is not set");
            }

            Int32 age = DateTime.Now.Year - DateOfBirth.Value.Year;

            if (DateOfBirth > DateTime.Now.AddYears(-age))
            {
                age--;
            }

            return age;
        }

        private String firstName;
    }
}
```

Class definition

A class is a construct that enables you to group together variables of other types, methods and events.
A class should declare an access modifier (*public*), must declare the *class* attribute and a class name (*Customer*).
Optionally, a class can by marked with the keyword *sealed*, which prevents the inheritance of this class or *abstract*, which prevents the creation of an instance of this class. Abstract is typically used for base classes.

Interfaces and inheritance

A class can implement Interfaces and inherit from **one** other class by placing a colon behind the class name, followed by the name of the base class and a comma separated list of interfaces.

Properties

A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field. A property has a getter and a setter with (different) access modifiers. If the getter and setter of a property has no body, this is an auto property. A Property that has a getter and setter with a body is called property with backing field.

Events

Events enable a class or object to notify other classes or objects when something of interest occurs. The class that sends (or raises) the event is called the publisher and the classes that receive (or handle) the event are called subscribers.

Override and virtual

It is possible to override class members (methods, properties …) of a base class with the *override* modifier if the base class member is marked with the *virtual* modifier. This is done to extend, or to change the behavior of a class member.

Methods

A method is a code block that contains a series of statements. It has an access modifier (*public*), a return type (*Int32*) or void if it does not return anything, a method name (*GetAge*), and optionally parameters that are passed to the method when called.

Using

The *using* directive defines namespaces of other classes that are used within your own class. In this example, the class *String* that is defined in the *System* namespace is used within the *Customer* class. In order to make the *String* class usable, the *Customer* class must declare the usage of the *System* namespace via *using System;*

Namespace

The namespace keyword is used to declare a scope that contains a set of related objects (class, enum …). You can use a namespace to organize code elements.

Constructors

A class must declare at least one constructor. The constructor is called when creating a new instance of the class with the *new* operator. If no constructor is defined, a default constructor is automatically created during compile time.
A constructor can have parameters to allow the caller to pass information when a new instance is created.

Access modifiers

Access modifiers are keywords used to specify the declared accessibility of a member or a type:

- *public*: Access is not restricted
- *protected*: Access is limited to the containing class or types derived from the containing class
- *internal*: Can only be used by all other classes within the same assembly
- *private:* Access is limited to the current class

Fields

Fields are variables declared at class scope. They must be marked with an access modifier and define a type and have a name. In addition they can have the keywords

- *readonly*: The fields cannot be changed at all, except within the class constructor
- *static*: The field is the same for all instances of this class
- *const*: During compile time, the value of the field is propagated where it is used. This means if you change the field, all code that uses this field must be recompiled.