Sql server, .net and c# video tutorial

Free C#, .Net and Sql server video tutorial for beginners and intermediate programmers.

Part 3 - Why and when should we use an abstract class

Suggested Videos:
Part 1 - Can you store different types in an array in c#
Part 2 - What is jagged array

This question can also be asked in a slightly different way
Give an example of where we could use an abstract class?

Let us understand when to use an abstract class with an example. An organisation has 2 types of employees

1. FullTimeEmployee

2. ContractEmployee


```
public class FullTimeEmployee
{
    public int ID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int AnnualSalary { get; set; }

    public string GetFullName()
    {
        return this.FirstName + " " + LastName;
    }

    public int GetMonthlySalary()
    {
        return this.AnnualSalary / 12;
    }
}
```

```
public class ContractEmployee
{
    public int ID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int HourlyPay { get; set; }
    public int TotalHoursWorked { get; set; }

    public string GetFullName()
    {
        return this.FirstName + " " + LastName;
    }

    public int GetMonthlySalary()
    {
        return this.HourlyPay * this.TotalHoursWorked;
    }
}

public static void Main()
{
    FullTimeEmployee fte = new FullTimeEmployee()
    {
        ID = 101,
        FirstName = "David",
        LastName = "Pie",
        AnnualSalary = 60000
    };

    Console.WriteLine(fte.GetFullName());
    Console.WriteLine(fte.GetMonthlySalary());

    Console.WriteLine("----------");

    ContractEmployee cte = new ContractEmployee()
    {
```

Design Patterns

SOLID Principles

ASP.NET Web API

Bootstrap

AngularJS Tutorial

jQuery Tutorial

JavaScript with ASP.NET Tutorial

JavaScript Tutorial

Charts Tutorial

LINQ

LINQ to SQL

LINQ to XML

Entity Framework

WCF

ASP.NET Web Services

Dot Net Basics

C#

SQL Server

ADO.NET

ASP.NET

```
        ID = 102,
        FirstName = "Sam",
        LastName = "Brooks",
        HourlyPay = 100,
        TotalHoursWorked = 40
    };

    Console.WriteLine(cte.GetFullName());
    Console.WriteLine(cte.GetMonthlySalary());
}
```

Notice that, we have designed FullTimeEmployee and ContractEmployee classes as stand-alone classes. Regardless of the employee type, all employees in the organisation are going to have ID, FirstName and LastName properties. We also compute the FullName of any employee by concatenating their FirstName and LastName. This means that, the above two classes (FullTimeEmployee & ContractEmployee) are related and there is, a lot of common functionality duplicated in them. The problem with this design is that, tomorrow, if want to introduce MiddleName property and if we have to include it in the computation of FullName, then we have to make the same change in both the classes. So code maintainability is going to be a big issue with this design.

To avoid these issues, we can move the common functionality into a base class. Using a common base class, we are going to get rid of the duplicated code.

The obvious next question is, How should we design the base class?
1. Should we design it as an abstract class
OR
2. Should we design it as a Concrete (Non abstract) class

Let's see what's going to happen, if we design the base class as a concrete class. All the common code is now present in BaseEmployee class.

```
public class BaseEmployee
{
    public int ID { get; set; }
```

GridView

ASP.NET MVC

Visual Studio Tips and Tricks

Dot Net Interview Questions

Slides

Entity Framework

WCF

ASP.NET Web Services

Dot Net Basics

C#

SQL Server

ADO.NET

ASP.NET

GridView

ASP.NET MVC

Visual Studio Tips and Tricks

Java Video Tutorials

Part 1 : Video | Text | Slides

Part 2 : Video | Text | Slides

Part 3 : Video | Text | Slides

Interview Questions

```
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public string GetFullName()
    {
        return this.FirstName + " " + LastName;
    }

    public virtual int GetMonthlySalary()
    {
        throw new NotImplementedException();
    }
}
```

Notice that, now the **"FullTimeEmployee"** class inherits from **"BaseEmployee"** class and has only the code that is specific to it.

```
public class FullTimeEmployee : BaseEmployee
{
    public int AnnualSalary { get; set; }

    public override int GetMonthlySalary()
    {
        return this.AnnualSalary / 12;
    }
}
```

Along the same lines, **"ContractEmployee"** class inherits from **"BaseEmployee"** class and has only the code that is specific to it.

```
public class ContractEmployee : BaseEmployee
{
    public int HourlyPay { get; set; }
    public int TotalHoursWorked { get; set; }

    public override int GetMonthlySalary()
    {
        return this.HourlyPay * this.TotalHoursWorked;
    }
```

```
}
```

So, with the above design we got rid of duplicated code, but we introduced another problem. Since "BaseEmployee " is a concrete (Non abstract) class, there is nothing stopping us from creating an instance of BaseEmployee class and using it. In the Main() method, someone could instantiate "BaseEmployee" class as shown below.

```
public static void Main()
{
    BaseEmployee be = new BaseEmployee()
    {
        ID = 101,
        FirstName = "David",
        LastName = "Pie",
    };

    Console.WriteLine(be.GetFullName());
    Console.WriteLine(be.GetMonthlySalary());
}
```

The above design is bad for 2 reasons

1. We only have 2 types of employees in our organisation - ContractEmployee & FullTimeEmployee. The developers should only able to instantiate ContractEmployee & FullTimeEmployee classes and not BaseEmployee class.

2. We get a run time error, if we invoke GetMonthlySalary() method on BaseEmployee class.

To get rid of the second issue, we can make the following modifications

1. Remove GetMonthlySalary() virtual method from BaseEmployee class

2. Remove "override" keyword from GetMonthlySalary() method in ContractEmployee and FullTimeEmployee classes.

With the above changes, we won't get the runtime error, but we would still be able to instantiate BaseEmployee class. So to prevent BaseEmployee class from being instantiated, let's mark it as an abstract class.

One more change is to introduce GetMonthlySalary() as an abstract method in BaseEmployee class. This will ensure that, all the classes that derive from BaseEmployee class,
1. Will either provide implementation for GetMonthlySalary() method
OR
2. The derived class will also be marked as an abstract class.

With the above changes the design looks as shown below.

```
public abstract class BaseEmployee
{
    public int ID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public string GetFullName()
    {
        return this.FirstName + " " + LastName;
    }

    public abstract int GetMonthlySalary();
}

public class FullTimeEmployee : BaseEmployee
{
    public int AnnualSalary { get; set; }

    public override int GetMonthlySalary()
    {
        return this.AnnualSalary / 12;
    }
}

public class ContractEmployee : BaseEmployee
{
    public int HourlyPay { get; set; }
    public int TotalHoursWorked { get; set; }
```

```
    public override int GetMonthlySalary()
    {
        return this.HourlyPay * this.TotalHoursWorked;
    }
}
```

So, in short, we would create an abstract class, when want to move the common functionality of 2 or more related classes into a base class and when, we don't want that base class to be instantiated.

8 comments:

Abhi kr September 28, 2013 at 1:44 AM

HiVenkat,
how can i execute the output of this query :
declare @sourceColName table(ID int identity(1,1),ColName nvarchar(256),compareCol char(1))
declare @TargetColName table(ID int identity(1,1),ColName nvarchar(256),compareCol

```
char(1))

declare @SourceTab table( ID nvarchar(256),Name nvarchar(256),Designation
nvarchar(256),joiningDate Datetime,sex char(1),test varchar(30))

declare @TargetTab table( ID nvarchar(256),Name nvarchar(256),Designation
nvarchar(256),joiningDate Datetime,sex1 char(1),test varchar(30))

insert into @sourceColName(ColName,compareCol) values('ID',1)
insert into @sourceColName(ColName,compareCol) values('Name',0)
insert into @sourceColName(ColName,compareCol) values('Designation',0)
insert into @sourceColName(ColName,compareCol) values('joiningDate',0)
insert into @sourceColName(ColName,compareCol) values('sex',0)
insert into @sourceColName(ColName,compareCol) values('test',0)

insert into @TargetColName(ColName,compareCol) values('ID',1)
insert into @TargetColName(ColName,compareCol) values('Name',0)
insert into @TargetColName(ColName,compareCol) values('Designation',0)
insert into @TargetColName(ColName,compareCol) values('joiningDate',0)
insert into @TargetColName(ColName,compareCol) values('sex1',0)
insert into @TargetColName(ColName,compareCol) values('test',0)

insert into @SourceTab values('1','Kripesh','lead','2013-07-24 17:37:30.087','M','abc')
insert into @SourceTab values('2','mohan','ssc','2014-07-24 17:37:30.087','M','abc')



insert into @TargetTab values('1','Kripesh','lead','2013-07-24 17:37:30.087','F','abc')
insert into @TargetTab values('2','mohan','lead','2013-07-24 17:37:30.087','M','eee')

-- scripts

declare @max int,@min int,@queryString varchar(8000)
select @queryString='select '

--select * from @sourceColName where compareCol=0

select @max=MAX(ID),@min=MIN(ID) from @sourceColName where compareCol=0

while (@min<=@max)
begin
```

```
set @queryString=@queryString+'stg.'+(select ColName from @sourceColName where compareCol=0 and ID=@min)+','
+'tgt.'+(select ColName from @TargetColName where compareCol=0 and ID=@min)+',+
(case when stg.'+(select ColName from @sourceColName where compareCol=0 and ID=@min
)+'=tgt.'+(select ColName from @TargetColName where compareCol=0 and ID=@min)+'
then "true" else "false" end ) Status ,'

set @min=@min+1
end

declare @sTabName nvarchar(256),@tTabName nvarchar(256),@finalSQL nvarchar(max)

set @queryString= substring(@queryString,0,len(@queryString))
set @finalSQL=''
set @sTabName='@SourceTab'
set @tTabName='@TargetTab'
set @finalSQL=((select @queryString+' from '+@sTabName + ' stg inner join '+ @tTabName + ' tgt on stg.' +
(select ColName from @sourceColName where compareCol=1)
+'=tgt.'+(select ColName from @TargetColName where compareCol=1)))

select @finalSQL
```

Thanks,
Abhi .

Reply

maran September 28, 2013 at 5:11 AM

So when would a base class required , Please a provide with an example.

Thanks

Reply

ESHK March 16, 2014 at 9:19 PM

its really a grt tutorial, but my doubt is ,Why we didn't use
Child Obj =new Abstract Class()
Why FullTimeEmployee=new FullTimeEmployee()
Why not FullTimeEmployee=new BaseEmployee();
Actually i am a newbie to C# please help me

Reply

Replies

khalid bin walid April 16, 2016 at 2:27 PM

in C#, we cannot create an instance from abstract class.

Reply

Habib Kawsar August 8, 2015 at 11:14 PM

No need to do that like FullTimeEmployee=new BaseEmployee(); as because both fulltime and parttime employee is already inheriting from base class

Reply

Anonymous September 10, 2016 at 10:46 AM

what will happen if base class is instantiated?

Reply

abhishek chauhan December 27, 2016 at 5:39 AM

It will throw an an error that cannot create an instance of the class which is marked as abstract. So , in short we cannot create an instance of that class which is defined as a abstract.

Reply

jeffry March 25, 2018 at 8:12 AM

Hi , i'm new using c# i would like to know why my solution has an error , i created a abstract class like this:
public abstract class Datos{
public string server{get; set};
public string user {get; set};
public string passwd {get; set};
}
i created 2 other classes :
Listar : Datos
{

}

Ejecutar: Datos
{
}
when i use console application it works very well but the problem is when i use windows form.
mean how to add data from textbox in the form,
example
server = textServer.text
I'm new using c# and vial net please can you help me?.

Reply

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)