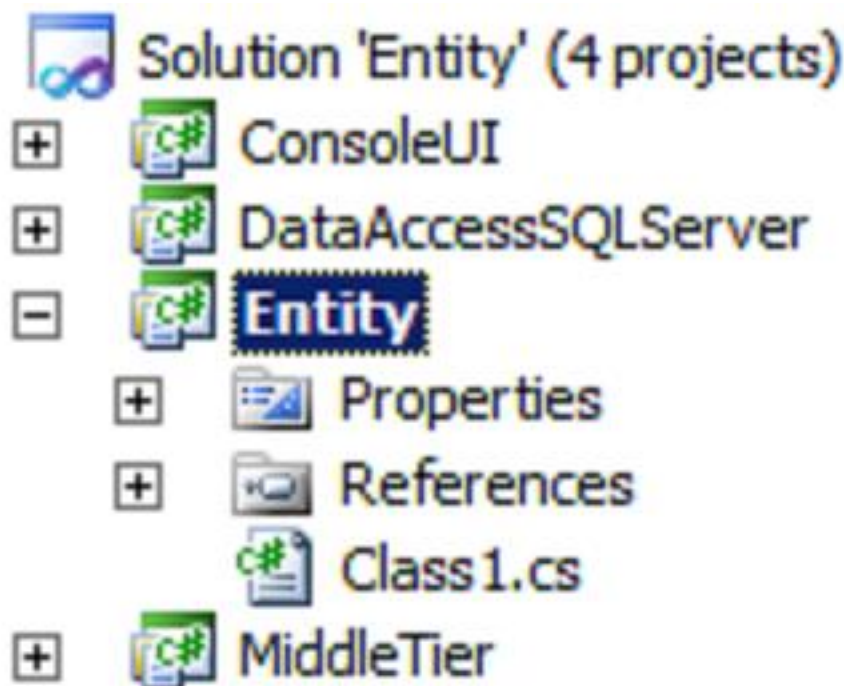


قسمت اول : تبادل داده ها بین لایه ها- قسمت اول

## روش دوم: (Uniform Entity classes)

روش دیگر پاس دادن داده ها، روش uniform است. در این روش کلاس های Entity ، یک سری کلاس ساده به همراه یکسری Property های Get و Set می باشند. این کلاس ها شامل هیچ منطق کاری نمی باشند. برای مثال کلاس CustomerEntity که دارای دو Property ، Customer Name و Customer Code می باشد. شما می توانید تمام Entity ها را به صورت یک پروژه ی مجزا ایجاد کرده و به تمام لایه ها رفرنس دهید.



```
public class CustomerEntity
{
    protected string _CustomerName = "";
    protected string _CustomerCode = "";
    public string CustomerCode
    {
        get { return _CustomerCode; }
        set { _CustomerCode = value; }
    }
    public string CustomerName
    {
        get { return _CustomerName; }
        set { _CustomerName = value; }
    }
}
```

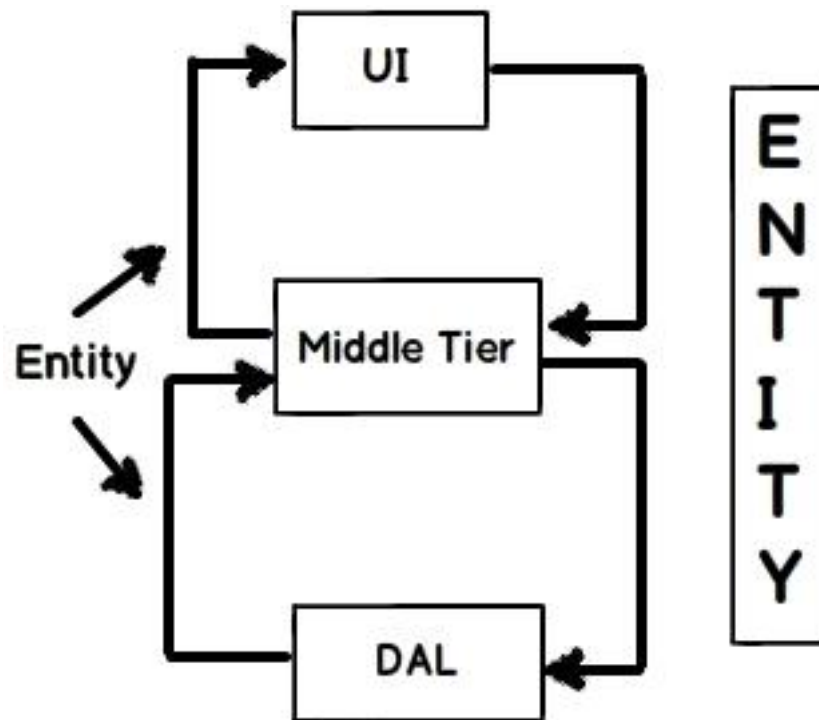
خوب، اجازه دهید تا از CustomerDal شروع کنیم. این کلاس یک Collection از CustomerEntity را بر می گرداند و همچنین یک CustomerEntity را برای اضافه کردن به دیتابیس. توجه داشته باشید که لایه Data Access وظیفه دارد تا دیتای دریافتی از دیتابیس را به CustomerEntity تبدیل کند.

```
public class CustomerDal
{
    public List<CustomerEntity> getCustomers()
    {
        // fetch customer records
        return new List<CustomerEntity>();
    }
    public bool Add(CustomerEntity obj)
    {
        // Insert in to DB
        return true;
    }
}
```

لایه Middle از CustomerEntity ارث بری می کند و یکسری operation را به entity class اضافه خواهد کرد. داده ها در قالب Entity Class به لایه Data Access ارسال می شوند و در همین قالب نیز بازگشت داده می شوند. این مسئله در کد ذیل به روشنی مشاهده می شود.

```
public class Customer : CustomerEntity
{
    public List<CustomerEntity> getCustomers()
    {
        CustomerDal obj = new CustomerDal();
        return obj.getCustomers();
    }
    public void Add()
    {
        CustomerDal obj = new CustomerDal();
        obj.Add(this);
    }
}
```

لایه UI هم با تعریف یک Customer و فراخوانی operation های مربوط به آن، داده ی مد نظر خود را در قالب CustomerEntity بازیابی خواهد کرد. اگر بخواهیم عمکرد روش uniform را خلاصه کنیم باید بگوییم، در این روش دیتای رد و بدل شده ی مابین کلیه لایه ها با یک ساختار استاندارد، یعنی Entity پاس داده می شوند.



مزایا و معایب روش uniform

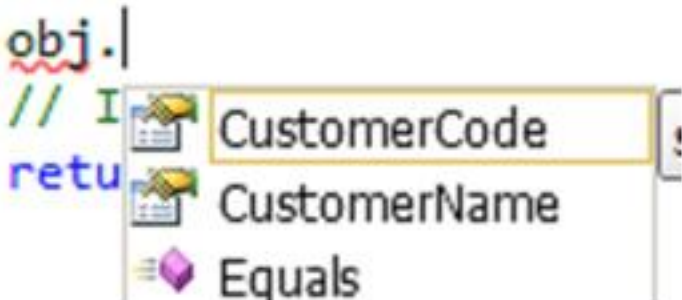
مزایا

• Strongly typed به صورت در تمامی لایه ها قابل دسترسی و استفاده می باشد.

```

public class CustomerDal
{
    public List<CustomerEntity>
    {
        // fetch customer record
        return new List<CustomerEntity>()
    }
    public bool Add(CustomerEntity obj)
    {
        // Insert
        return true
    }
}

```



• به دلیل اینکه از ساختار عمومی Entity استفاده می‌کند، بنابراین فقط یکبار نیاز به تبدیل داده‌ها وجود دارد. به این معنی که کافی است یک بار دیتای واکنشی شده از دیتابیس را به یک ساختار Entity تبدیل کنید و در ادامه بدون هیچ تبدیل دیگری از این Entity استفاده کنید.

### معایب

• تنها مشکلی که این روش دارد، مشکلی است به نام Double Loop. هنگامیکه شما در مورد کلاس‌های entity بحث می‌کنید، ساختارهای دنیای واقعی را مدل می‌کنید. حال فرض کنید شما به دلیل یکسری مسایل فنی دیتابیس خود را Optimize کرده اید. بنابراین ساختار دنیای واقعی با ساختاری که شما در نرم افزار مدل کرده‌اید متفاوت می‌باشد. بگذارید یک مثال بزنیم؛ فرض کنید که یک customer دارید، به همراه یکسری Address. همان طور که ذکر کردیم، به دلیل برخی مسایل فنی (denormalized) به صورت یک جدول در دیتابیس ذخیره شده است. بنابراین سرعت واکنشی اطلاعات بیشتر است. اما خوب اگر ما بخواهیم این ساختار را در دنیای واقعی بررسی کنیم، ممکن است با یک ساختار یک به چند مانند شکل ذیل برخورد کنیم.

