Immutable object:

Objects once created/loaded cannot be modified/changed by anyways externally or internally

Eg: master data

Currency data, country data, region data,

Singleton object:

Webconfig

3 step process

1. All setters need to be removed only getters possible
2. Use constructor to initialize the data to the class.
3. To the class modifying the logic make variable as read-only

Used for static data: singleton, cache, master data.

Lazy Loading:

On demand loading . Load data only when it is requested, SO basically we delay the loading of object until certain point is reached.

Useful in entity relation when relation data is loaded only when it is required don’t load everything at once

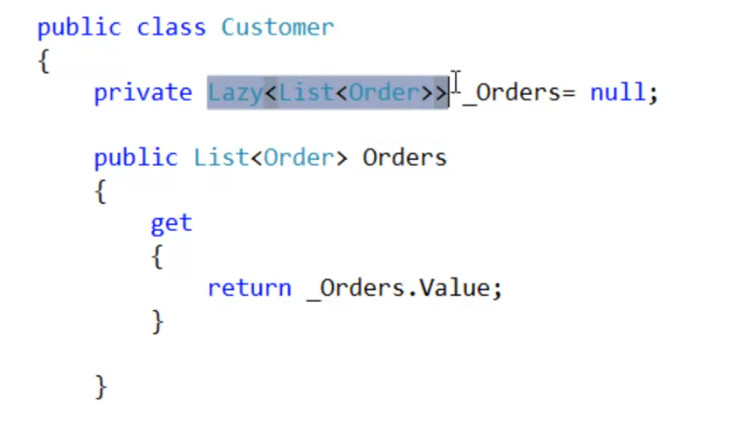
Aproach1:

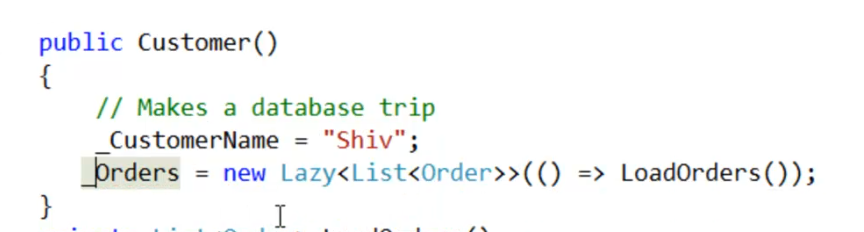
Define one private property one public property to load data load data to private property only when the call is made

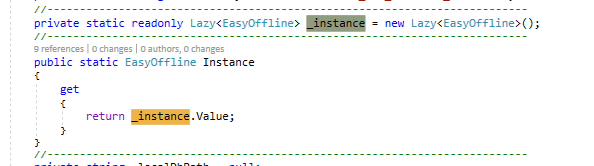
Aprocah2:

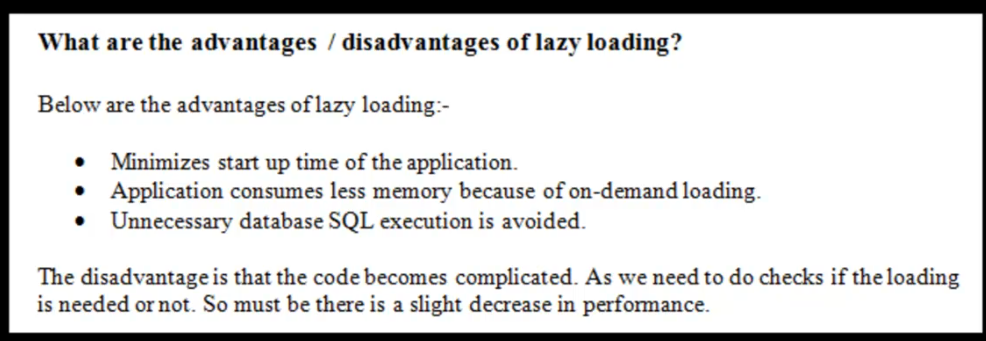
Use Lazy keyword from .net library. Define property as lazy and in constructor specify which method to use to load the data.

So define property as lazy









**IOC: inversion of control:**

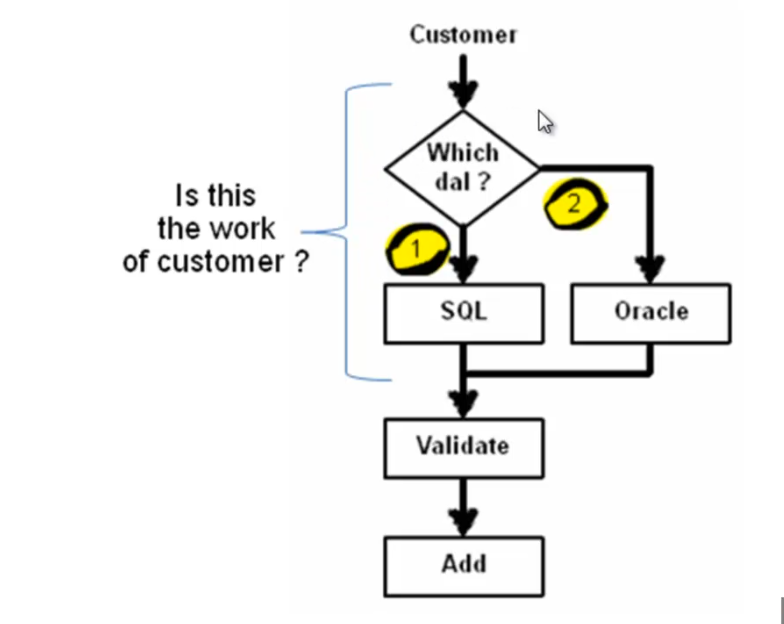
Pogroming technique in which unconcerned program logic is delighted to some other entity.

Eg: customer object internally has Db object so we need to decide on which Data access layer to use before making the operation on customer object.

Here making a decision on which DAL to use is not responsibility of the customer object.

So this logic we have to move out of this entity.

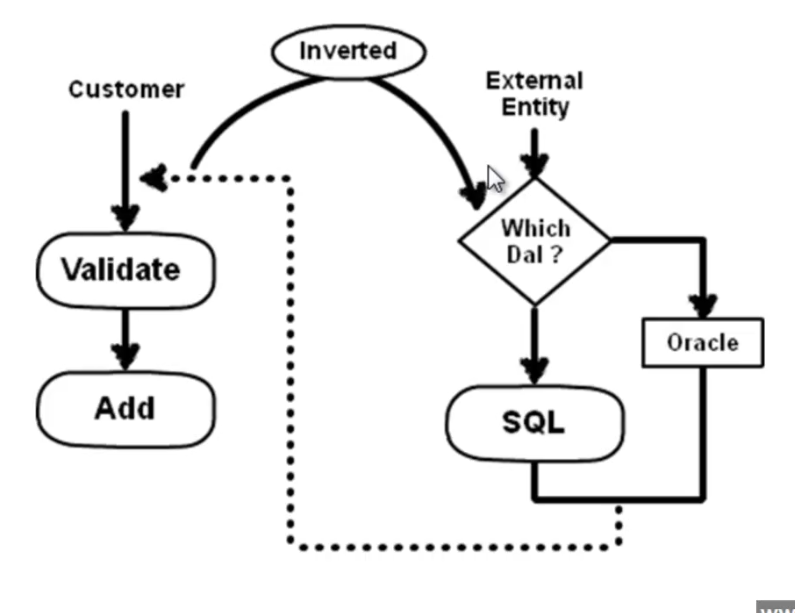
Problem:



Solution:

Here control is inverted to external entity:

Code is loosely coupled.



**Fluent interfaces and method chaining:**

Deals with more into consumer part of the class or application when is you develop class in more readable, discoverable and simple

Method chaining is a design technique where in all the methods return an object. And we can club all these methods together to forma single statement.

Where code is written like as if you are writing some English statements

Here

Define class

And make a wrapper class which instantiates above class and then write a methods that takes parameter and each method return current object so that next method is called with the same instance and return void from last method to be called so tat there is no method chaining further possible on that object.

**Design pattern:**

Documented , tried and tested solution for the recurring problems.

Category or areas of problem in application

1. Creational : centralize the process of object creation delegate this operation to single entity
2. Structural: Application structure where in how different modules entity are linked
3. Behavioural: Design application in such a way that change to one behaviour will not affect the entire application.