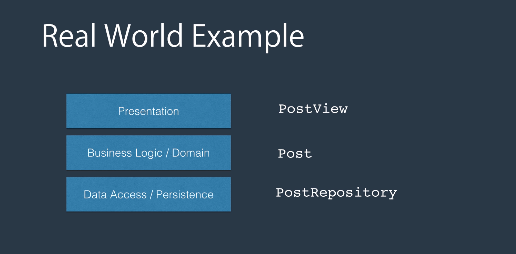
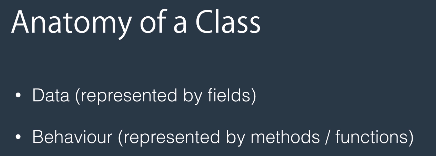
Class:

Building block of the application

Each responsible for one behaviour

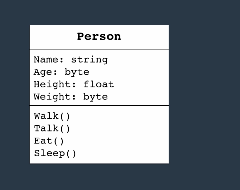


Type defined by developer responsible for some functionality



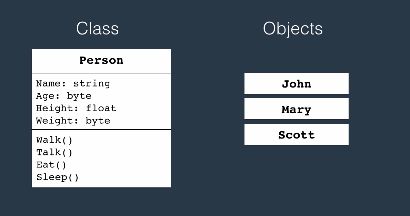
UML notation

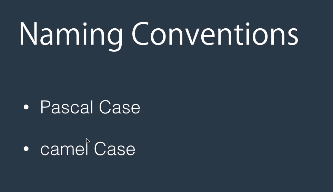
Graphical representation of class



Object:

Instance of the class present in the memory





Members of the class

Instance members:

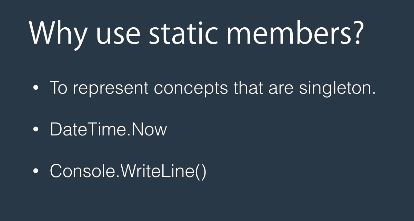
Access though instance of the class

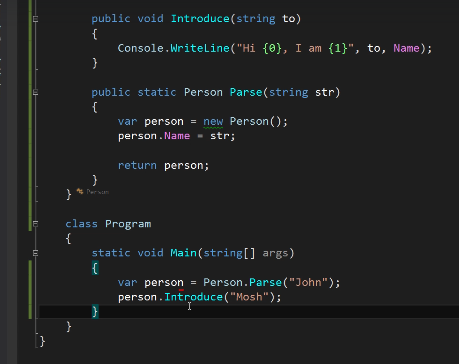
Static members:

Accessed directly from the class.

Use of the static member:

Internally it uses the design patterns singleton i.e At given point of time only one instance of the object will be in memory.





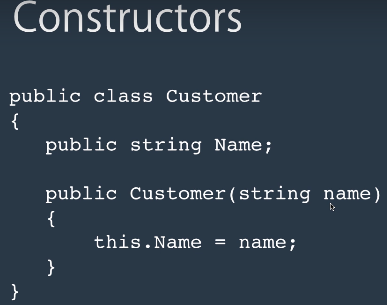
Constructor:

Name same as class name

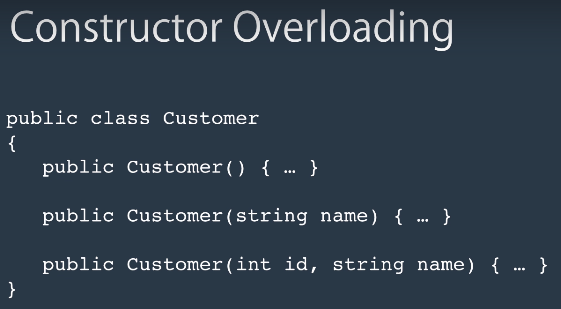
No return type

Initialize object at the time of creation

Always added default constructor inside IL if not define and initialize the primitive types to its default values



Constructor overloading

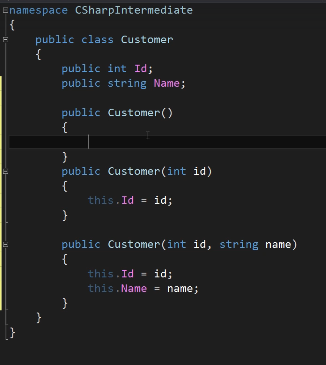


In real world scenarios never put the fields with public modifier.

Snippet for constructor

Ctor +tab

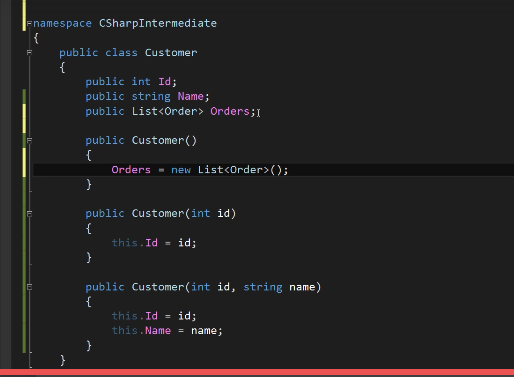
If you add a other version of contractor then it is not possible to use the create an object with default constructor we have to add a default constructor.



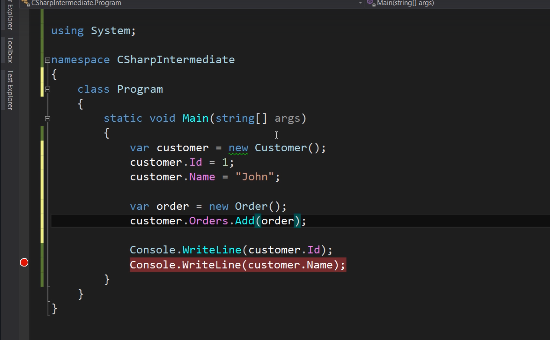
But we need not always use constructor initialize fields inside the class. Useful in a scenario where the object needs to be initialized in immediate basis.

Example for this is

Says I have customer class and orders class where in customer have list of orders. So order collection need to be initialized when customer object is initialized.

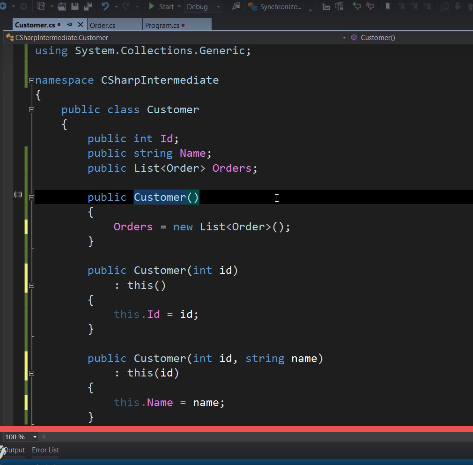


This code will crash



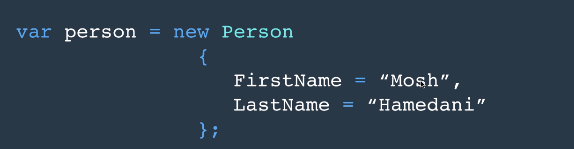
Null reference exception for the orders list.

Its also possible to call other version of the constructors in the same class. By using this key word and overloaded version of it as shown below.



Object initialization (avoiding the constructor)

Because as fields increases in the class it will end up adding too many constructor for the different combination

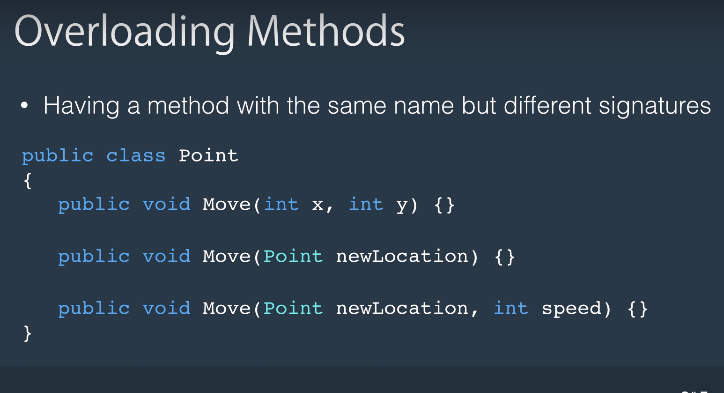


**Methods:**

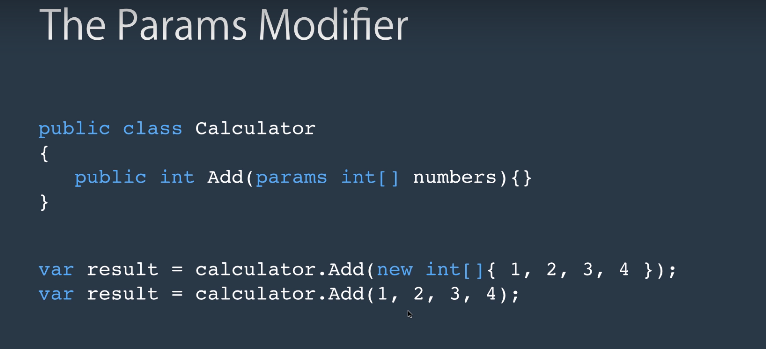
**Signature:** name, number and type of parameter

**Method overloading**

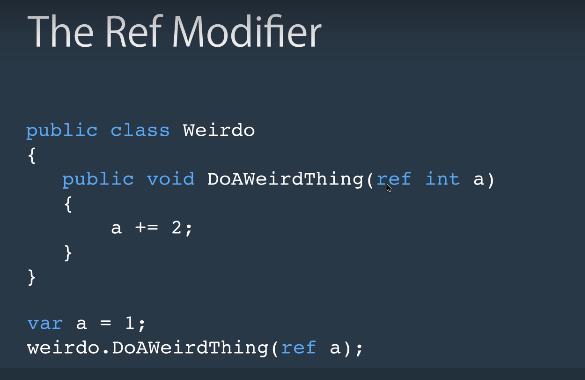
Method with same name but different signature.



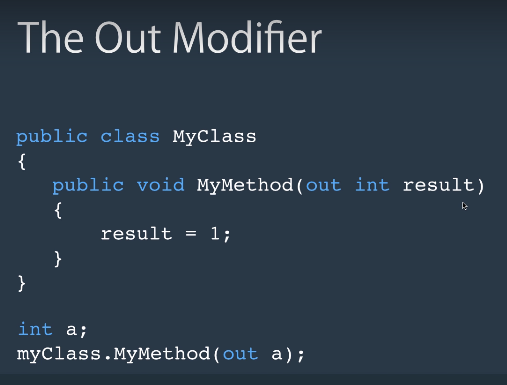
**Params parameter: To pass variable number of argument to single variable**



**Ref modifier:** pass by value method

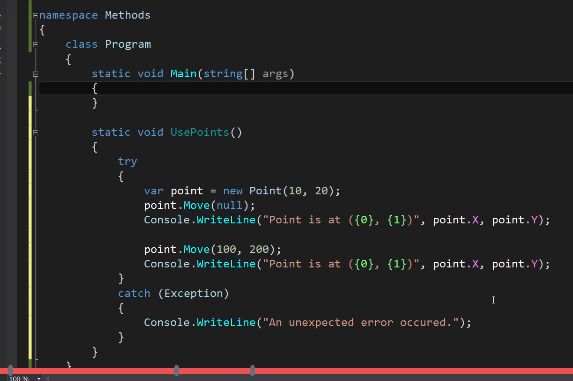


**Out:** used to return the multiple values from the class.



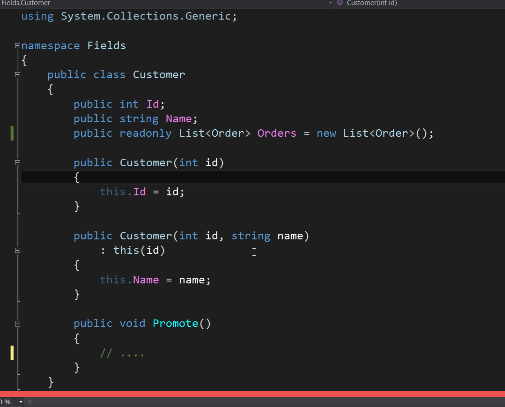
Writing code with all exception handling is called defensive programming. Such code is called robust code

This is don with the help of exception handling try catch block



Fields:

Data fields representing the class data are called fields.



Object level field which can be initialized only at the time of declaring it or at the constructor level. This will increate the robustness of your code by blocking the code from resetting the value anywhere in the code.

**Access modifiers:**

Way to provide the control access to the class and its fields and behaviours.

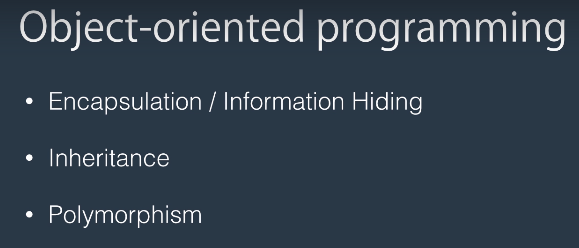
Public

Private

Protected

Internal

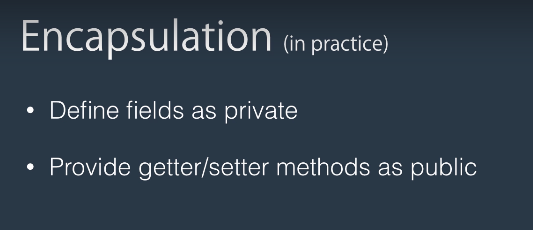
ProtectedInternl



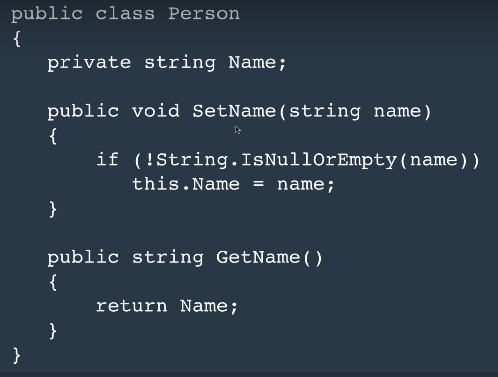
Encapsulation:

Hiding the information about the class from other class. We don’t want to tell the class about the other class how it does the things.

Encapsulate the data in terms of fields and behaviour in terms of method



Using methods to i



Properties:

Class member that encapsulate Getter/Setter inside the class for accessing a field of the class.

Why:

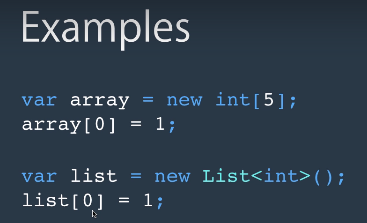
Generate getter setter with minimum and well structure code.

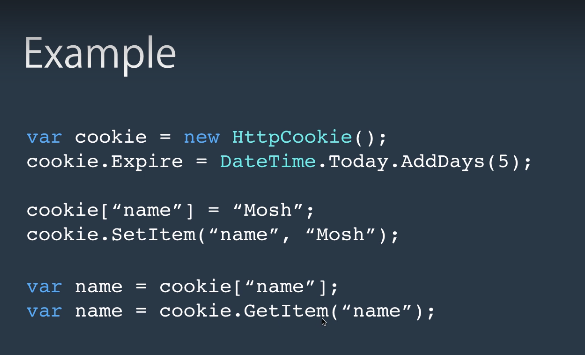
Property without any code only with getter, setter. When compiler sees it. Internally generates the private field



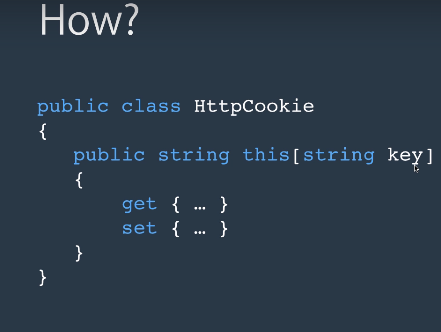
Indexers:

Way to access the elements in the class that represents a list of values.

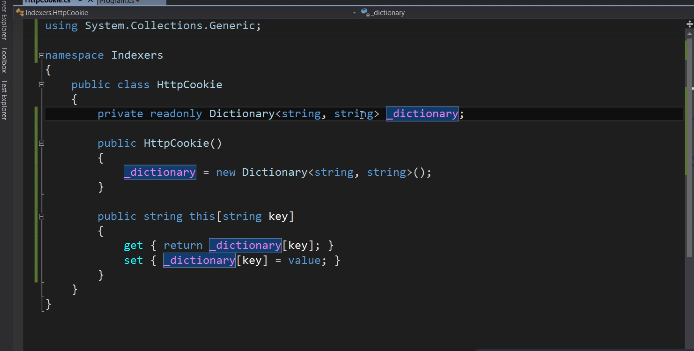




Similar to the property in the class



Class which has the semantic of the collection user indexer



**Indexer:**

Just like property used to access the data fields

Will not allocate memory

Provides the way to access the data member.

Can access multiple data members

Can have either one or both the accessor

Instance member so cant be static.

It need not be associated with any fields or property because as long as it returns some value its safe.

**Declare indexer:**

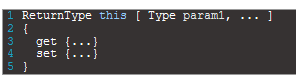
Does not have any name

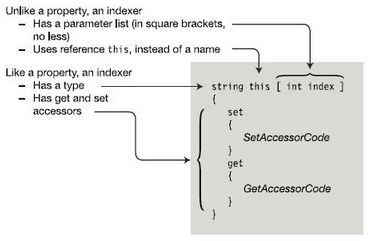
In the place of name use this keyword

Return value

And parameter list in between the []

Should have at least one parameter.

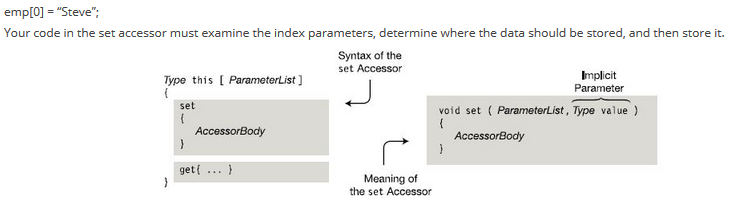




Indexer as set accessor:

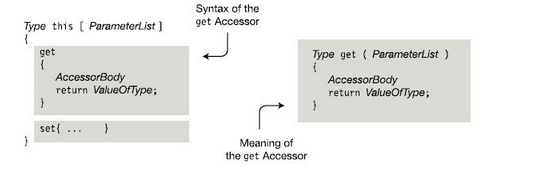
When target of assignment set accessor is called.

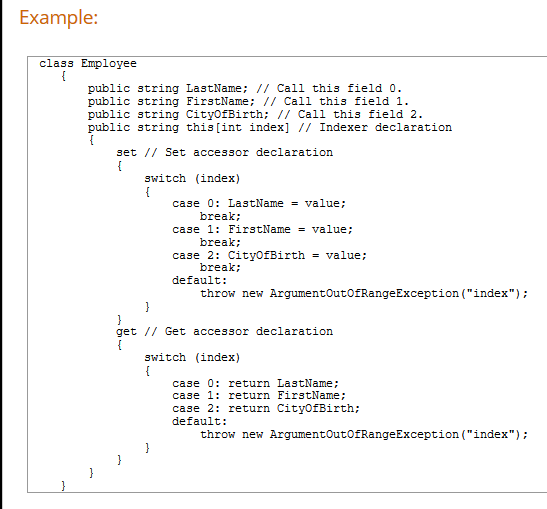
Receives Implicit parameter name value which holds the data to be saved. Index parameter says where to be saved.



Indexer as get accessor:

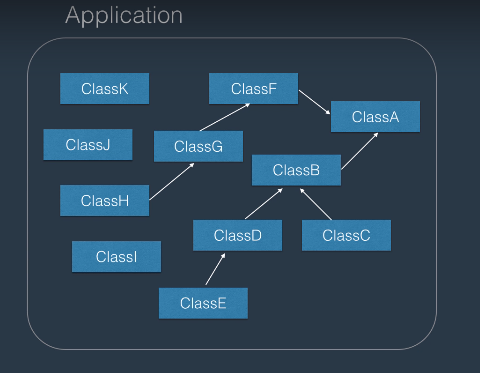
When we try to get a value from the property Get accessor is called.





Class coupling:

Measure of how interconnected class and subsystem are.

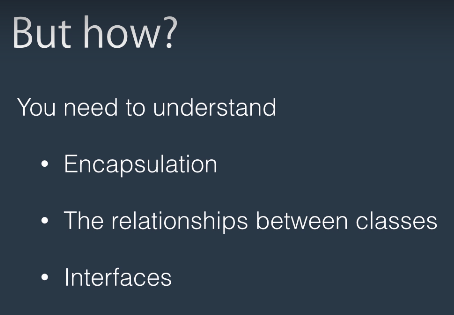


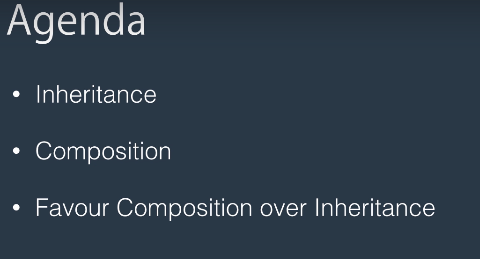
Tight coupling:

Strong dependency between the class.

Loos coupling:

There is no complete dependency between the class.

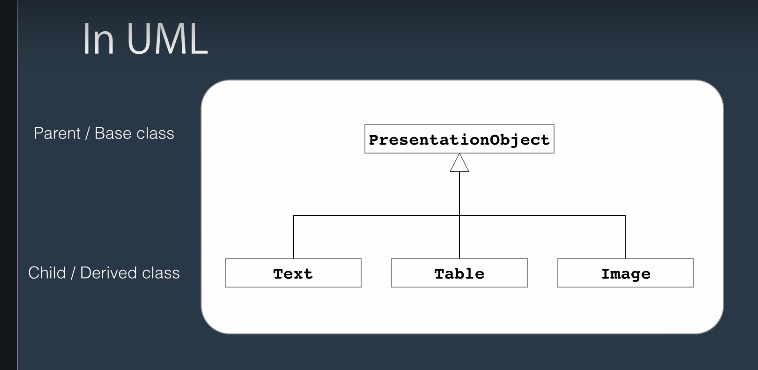




UML: Unified modelling language used to represent the class in graph

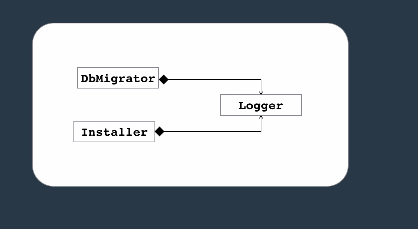
Inheritance:

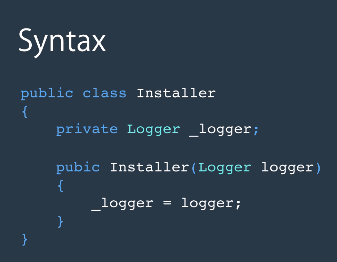
1. Relationship between the classes that allows one class to inherit the code from the other
2. It’s a Is-A relation ship
3. Eg: john is a person, car is a vehicle.

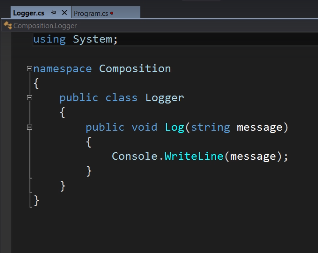


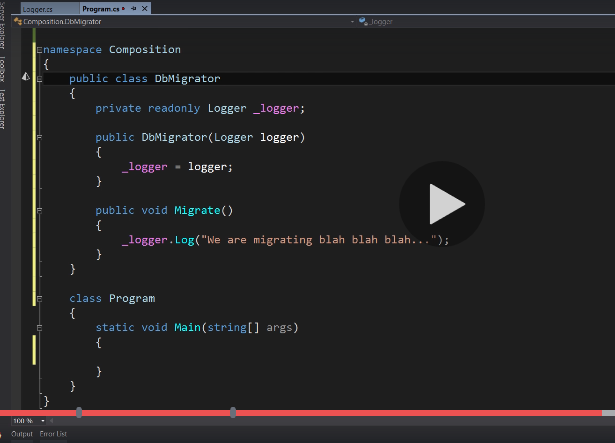
Composition:

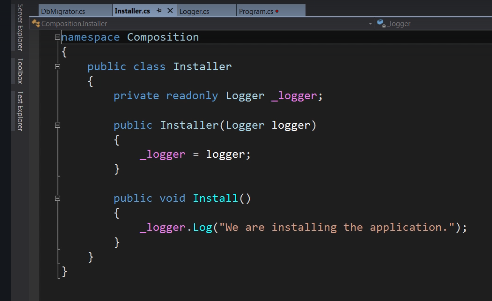
1. Kind of relation between the class where one class contains another class inside it.
2. Has a relation ship
3. Eg: Car has engine.

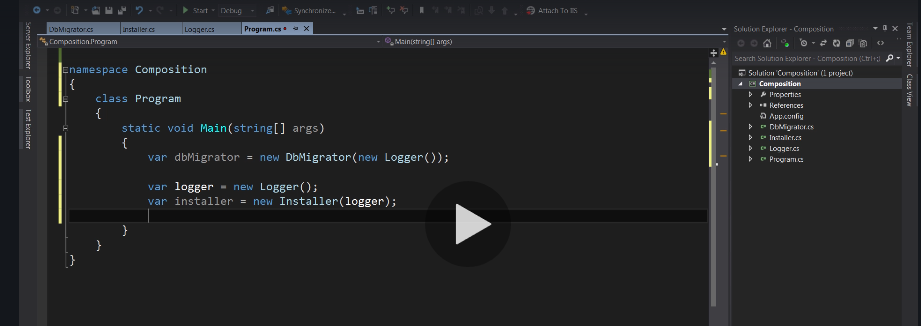




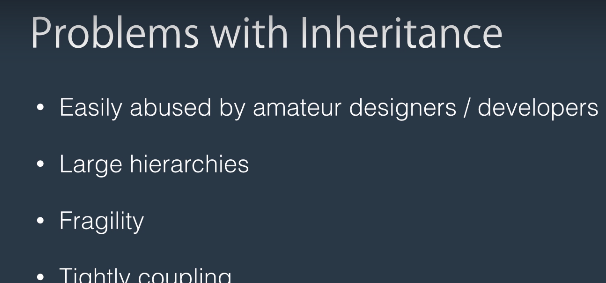




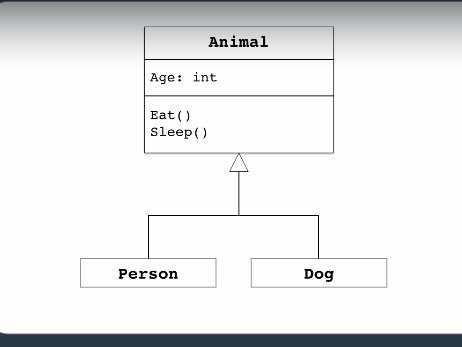




Favour of composition over inheritance:

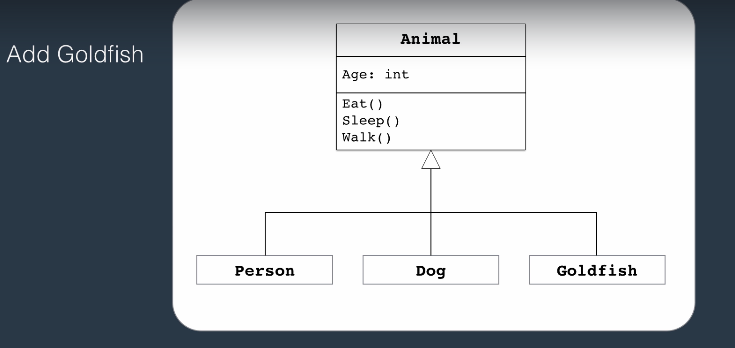


EG

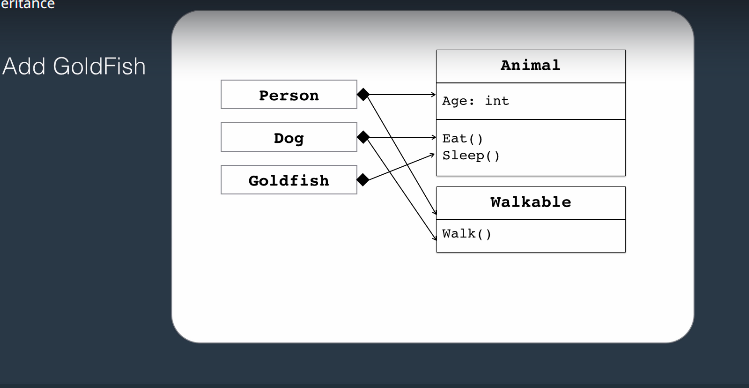


A method assed called Walk();

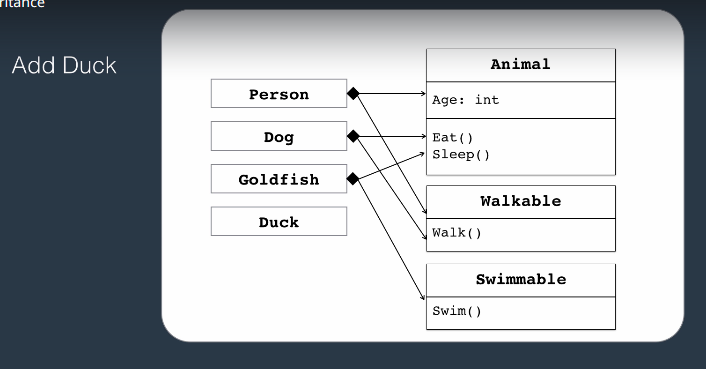
Both class need to be changed and recompiled to implement that method.

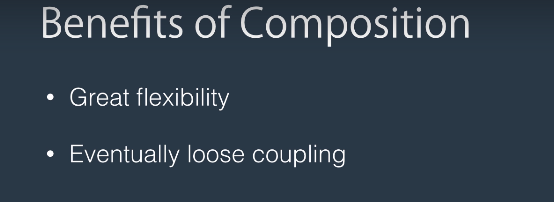
Suppose if I add another class to the hierarchy its clear that the gold fish cannot walk. SO to support this may be I have to add another intermediate class called mammal that implements animal and from mammal we need to inherit persona and dog

Solution is with composition:



Add two separate class and then include that inside the actual class.





Access modifiers:

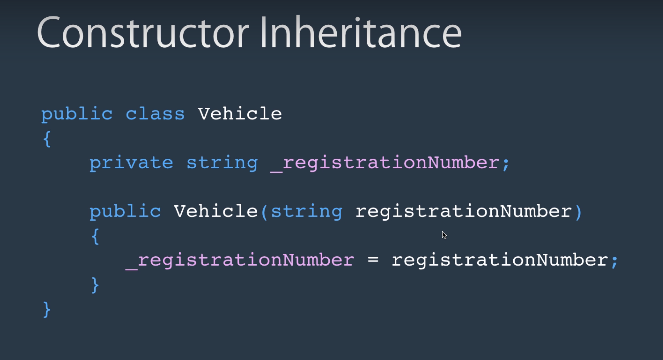
Limited visibility from outside lot of things happening inside

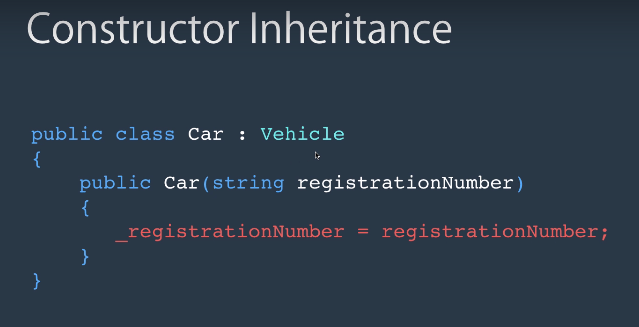
If the code become more visible it lead to lot of dependency and tight coupling between the class so always try to reduce the dependency by limiting the accessibility.

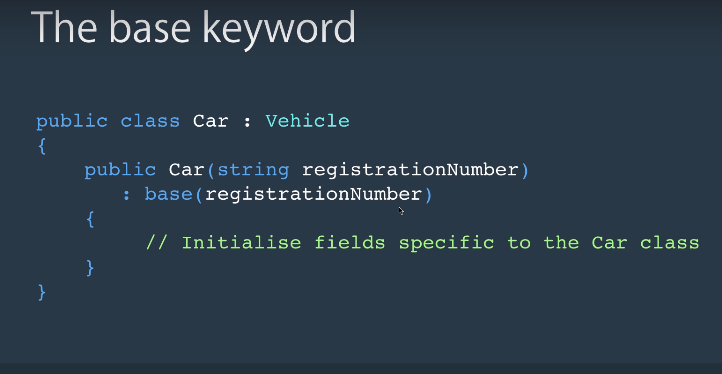
Constructors and inheritance:

Base class constructor are always called first

Base class constructors cannot be inherited.

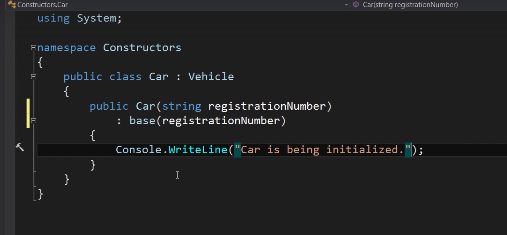




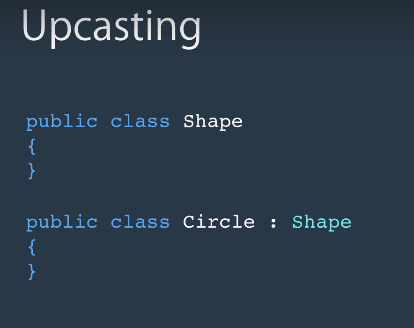


In class hierarchy if you define a constructor for the child class without having constructor for parent class it ill lead to an error and compilation breaks.

Because the CLR is not aware like how to make parent class object when child object is initiated.

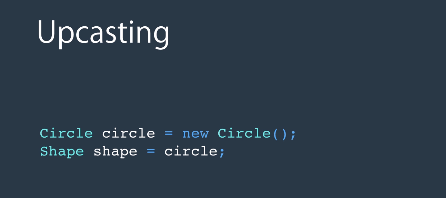


Upcasting and down casting:



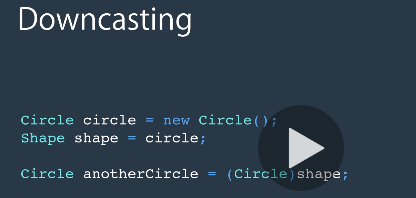
Converting child class to its parent class is called as up casting.

When casting is doe both object is referring same location but with different views.



Down casting:

Converting parent class to child class object



Casting can be done only to the respective compatible types or else it will throw an exception

**As key word:**

Conversion without throwing exception

**Is key word:**

Used for type checking.

**Boxing and unboxing:**

**Value types:**

Are stored in stack, All primitive types saved here(int, float, bool etc)

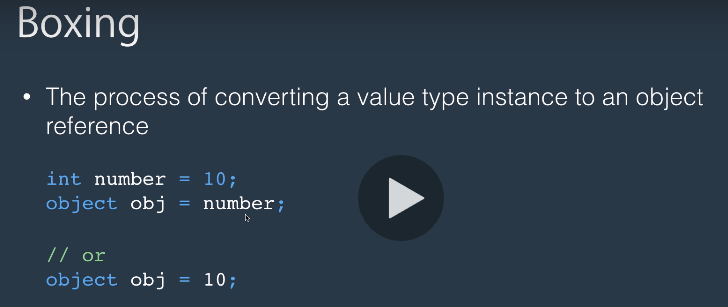
Short lifetime once the scope is lost memory will be released.

**Reference type:**

Values will be save don heap and have longer life time Eg. Class, string , Array etc.

**Boxing:**

Converting a value type to reference type is called as boxing.



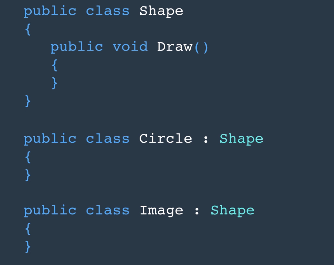
**Unboxing:**

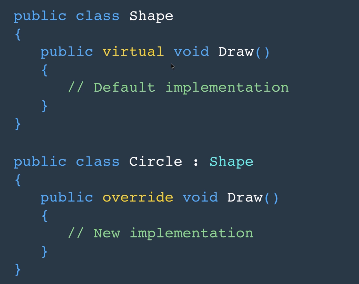
Converting reference type back to value type is called unboxing.

Third pillar of oops Polymorphism:

Overriding: Modifying the implementation of the inherited method

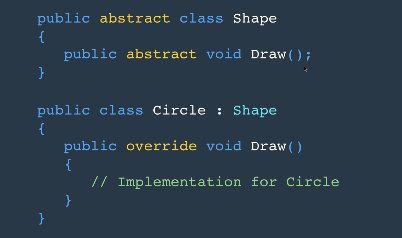
Say in the below example the way we draw different shapes uses different code so we need to change the way in which the shape is drawn.





Abstract modifier:

Indicates class or the member is missing implementation



Any member is defined as abstract we need to make class abstract

Cannot include implementation

If member abstract class should be abstract

Override all methods in child class

Use Provide some common implementation by which you are forcing the other developer to follow your implementation.

Sealed class:

Prevents derivation of the class or inheritance from the class

Can be applied to class or method.

Methods marked with override can be sealed so that no more overridden can be done on this method

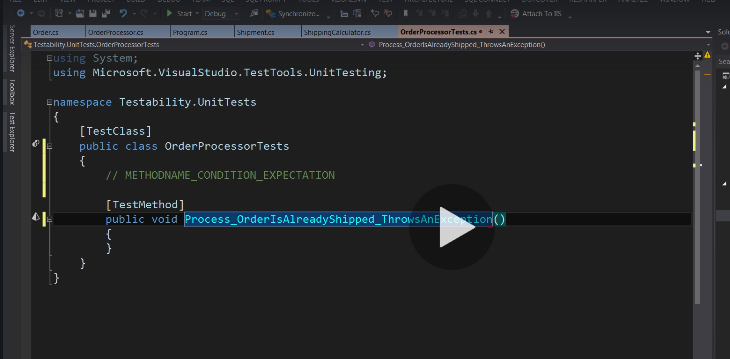
Interface:

Language construct similar to class but syntactically different that of the class

To build loosely couple application

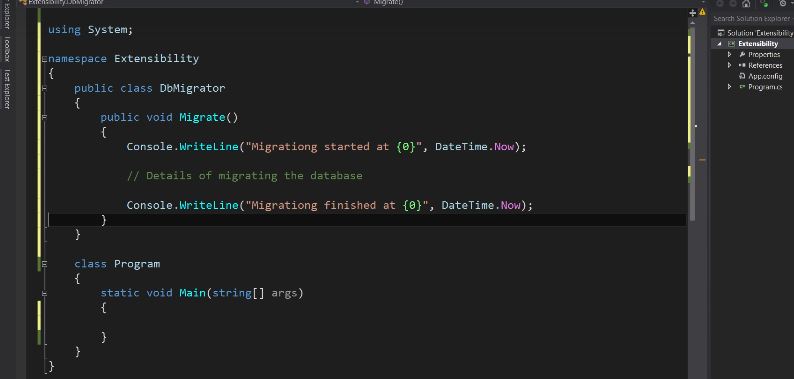
Interface method should be public in the derived class

No access modifier for the method in interface definition.



Interfaces Extensibility:

Db migration:



For now we write the log to console but in future if you want to write the log to some file or Db this code need to be changed.

This can be acived with interface extensibility.



Interface and inheritance:

Interface is nothing to do with inheritance

Delegates:

Type safe function pointers that holds the reference to the function.

Signature of the delegate point to the signature of the function that it points else throws compiler error

Type safe pointer because holds the reference of the function signature that matches with its signature.

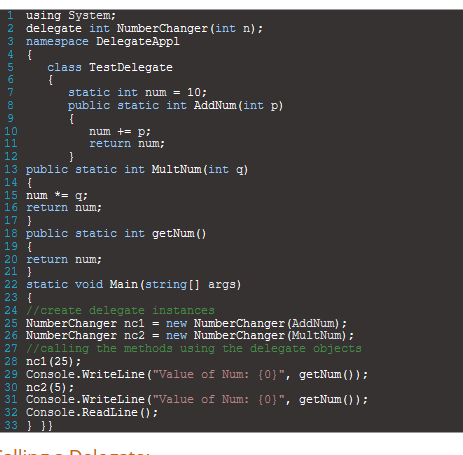
Implicitly derived from System.Delegate

Declare a deligate

delegate <return type> <delegate-name> <parameter list>

public delegate int TestDelegate (string s);

Once the delegate is declared. You need to create an object of the delegate using new key word to associate the method with delegate.



Once the delegate is assigned to a method. Its passed to other code that calls delegate

Next we need to call the delegate this can be done with the help of

With this we can call the method using delegate

Multicast delegate:

Delegate method that point to more than one method is called as multicast delegate

* + or += to register a method with the delegate
* – or -= to unregister a method with the delegate

Methods are invoked In the same order in which it is registered to the invocation list

If there are multiple methods with one as void return value and other has int value

Return value will be the value returned from the last ex3ecuted method.