**ASSIGNMENT DOCUMENT**



ARPIT PANDEY

(ASET), TAAZAA

**Properties**

Properties are combination of variable and a method where get and set are two methods.

* Get is used to return value and known as getter.
* Set is used to assign value to variable and known as setter.
* When we use only get and remove set then we say that method is of read only type.
* When we use only set and remove get then we say that method is of setter type or write only type.
* Access modifier of properties should always be narrower than that of method.
* Never compute value of getter for which we have setter it may lead to stack overflow.

Normally we have three type of property initializer: -

1. Automatic property initializer
2. Inline property initializer
3. Object property initializer

In object property initializer we have two syntaxes

* With curly braces
* With round braces

E.g. code:

Automatic: public int age {get; set ;}

Inline: public string name {get => name; set => name = value ;}

With curly braces: person p = new person {age = 29 ;}

With round braces: person p = new person () {name =”foo”, age = 9,}

**Keywords**

**Virtual:** This keyword is added in the parent or base class so that the class inheriting parent having same method name can be able to override them. Child having keyword override in them can be able to do so.

**Override:** This keyword is added in the child class so that method of the base class should be overridden.

**New:** This keyword is added in the child class if we have assigned base class with virtual keyword and don’t want method to get overridden instead, we want to have child to have its own method with the same name that parent is having.

**Sealed:** This keyword is added in the child class so that hierarchy of get overridden should get stopped at that particular child class and no other class should not be able to inherit that class.

Pseudo code:

Namespace Polydemo

{  
 Public class ClassA   
 {

**//ClassA is declared with keyword virtual to allow child to override method display ()**  
 public virtual void display () {

Console.WriteLine ("A class Method is called");  
 }  
 }

Public class ClassB: ClassA   
 {

**// ClassB overrides ClassA method display () using keyword override.**

**// ClassA a2 = new ClassB ();**  
 public override void display ()  
 {  
 Console.WriteLine ("B class method overrides class A method");  
 }  
 }

Public class ClassC: ClassB   
 {

**// ClassC overrides ClassA method display () but further inheritance will not be there due to sealed keyword.**

**// ClassA a1 = new ClassC ();**  
 public sealed override void display ()  
 {  
 Console.WriteLine ("C has own Aclassmethod");  
 }  
 }

Public class ClassD

{

**// ClassD has its own method display ()**

Public new void display ()

{

Console.WriteLine ("C has own Aclassmethod");

}

}

}

**Constructor**

Constructor is automatically get called when an object of class is created. Constructor is used to initialize the value of data members of the same class for which it is invoked or get called.

* If we are creating parameterized constructor of parameterless method then we have to explicitly define a parameterless method.
* Static constructor must be parameterless and cannot be overloaded.
* Class can have any no of constructor but one class can have only one static constructor.
* Constructor don’t have return type.
* Non static fields cannot be accessed inside static constructor.

There are normally following types of constructor:-

1. Default constructor
2. Instance constructor
3. Static constructor
4. Private constructor
5. Copy constructor
6. Parameterless constructor

**Default Constructor:** when we call object of class at that time by default one constructor is get created that is called default constructor.

Pseudo code:

Person p1 =new person ();

**//when object p1 will be created by default a constructor will be created.**

**Instance Constructor:** when we call constructor by passing parameter through that constructor then we call that constructor instance constructor.

Pseudo code:

Public person (string name, int age)

{

This.name = name;

This.age = age;

}

**Static Constructor:** when we call function using static keyword that constructor called static constructor. Static constructor is called just before the 1st object comes in count. Static constructor is used to update the field before it is assigned.

Pseudo code:

Public static string product (string name, int age): base (name, age)

**// calling the base class constructor to pass the value in static class child constructor.**

**Private Constructor:** when we create constructor using private keyword then we call it as private constructor. If we don’t want to create the object of the class then we use private constructor.

Pseudo code:

Private void person ()

**// we declare this constructor as private so that it has no scope out of this assembly so it will give compile time error when compiled.**

**Copy Constructor:** when we create a constructor that copies data of one object into the other object then we call it as copy constructor. Copy constructor is parameterized constructor with same parameter type as of class.

Pseudo code:

Public person (person p)

{

Name = p.name;

Age = p.age;

}

Person p2 =new person (p2);

**// this constructor copies the value of one object to another object.**

**Parameterless Constructor:** when we create constructor without any parameters then we call it as parameterless constructor. If we override the parameterless constructor with instance constructor still we have to declare parameterless constructor else complier will give error about that.

Pseudo code:

Public void person () {}

**// we generally create this constructor when we have instance or parameterized constructor which overrides the parameterless constructor, we declare it again to avoid compile time error.**