**Lesson01 Proto Type Design Pattern**

**Notes:-**

**1-prototype: when it’s easier to copy an existing object to fully initialize new one.**

**When to use it**

**When to use:-**

**1-complicated objects (e.g. cars) aren’t designed from scratch**

**2-an existing (partially or fully constructed) design is a prototype**

**3-we make copy (clone) the prototype and customize it (apply deep copy support)**

**4-make cloning convenient (vis factory)**

**Lesson02 Why ICloneable is bad**

**Notes:-**

**1-by default ICloneable is using only for shallow copy which means that it will refer the two reference variable to the same instance , so its bad to use IClonable in deep copy**

**using System;**

**namespace ICloneablePro.Models{**

**public class Address : ICloneable{**

**public string StreetName { get; set; }**

**public int HouseNumber { get; set; }**

**public Address(string streetName,int houseNumber){**

**if (streetName == null){throw new ArgumentNullException(paramName: nameof(streetName)); ;}**

**StreetName = streetName;**

**HouseNumber = houseNumber;}**

**public override string ToString(){**

**return $"{nameof(StreetName)} : {StreetName} , {nameof(HouseNumber)} : {HouseNumber}";}**

**//we inherit ICloneable in Person and Address class and it will apply shallow copy**

**public object Clone(){return new Address(StreetName, HouseNumber);}}}**

**using System;**

**namespace ICloneablePro.Models{**

**public class Person : ICloneable{**

**public string[] Names { get; set; }**

**public Address Address { get; set; }**

**public Person(string[] names, Address address){**

**if (names == null){throw new ArgumentNullException(paramName: nameof(names)); ;}**

**if (address == null){throw new ArgumentNullException(paramName: nameof(address)); ;}**

**Names = names;**

**Address = address;}**

**public override string ToString(){**

**return $"{nameof(Names)} : {string.Join(",",Names)} , {nameof(Address)} : {Address}";}**

**//we inherit ICloneable in Person and Address class and it will apply shallow copy**

**public object Clone(){return new Person(Names, (Address)Address.Clone());}}}**

**using ICloneablePro.Models;**

**using static System.Console;**

**namespace ICloneablePro{**

**class Program{**

**static void Main(string[] args){**

**var john = new Person(new[] { "John", "Smith" },new Address("London St",123));**

**WriteLine(john.ToString());**

**//this way is inefficent**

**//var jane = john;**

**//jane.Names[0] = "Jane";**

**//we see that it will make shallow copy which make the two refernces**

**//refer to same instance object**

**var jane2 = (Person)john.Clone();**

**jane2.Address.HouseNumber = 2345;**

**WriteLine(jane2.ToString());**

**ReadLine();}}}**

**Lesson03 Copy Constructor**

**Notes:-**

**1-another approach to apply deep copy is to make overload constructor that accept instance of the target instance we want to clone and make overloaded constructor foreach internal instance property (not recommended)**

**using System;**

**namespace CopyConstPro.Models{**

**public class Address{**

**public string StreetAddress, City, Country;**

**public Address(string streetAddress, string city, string country){**

**StreetAddress = streetAddress ?? throw new ArgumentNullException(paramName: nameof(streetAddress));**

**City = city ?? throw new ArgumentNullException(paramName: nameof(city));**

**Country = country ?? throw new ArgumentNullException(paramName: nameof(country));}**

**//to apply deep copy we pass the instnace and assign value with apply initialize of sub class**

**public Address(Address other){**

**StreetAddress = other.StreetAddress;**

**City = other.City;**

**Country = other.Country;}**

**public override string ToString(){**

**return $"{nameof(StreetAddress)}: {StreetAddress}, {nameof(City)}: {City}, {nameof(Country)}: {Country}";}}}**

**using System;**

**namespace CopyConstPro.Models{**

**public class Employee{**

**public string Name;**

**public Address Address;**

**public Employee(string name, Address address){**

**Name = name ?? throw new ArgumentNullException(paramName: nameof(name));**

**Address = address ?? throw new ArgumentNullException(paramName: nameof(address));}**

**//to apply deep copy we pass the instnace and assign value with apply initialize of sub class**

**public Employee(Employee other){**

**Name = other.Name;**

**Address = new Address(other.Address);}**

**public override string ToString(){**

**return $"{nameof(Name)}: {Name}, {nameof(Address)}: {Address}";}}}**

**using CopyConstPro.Models;**

**using static System.Console;**

**namespace CopyConstPro{**

**class Program{**

**static void Main(string[] args){**

**var john = new Employee("John", new Address("123 London Road", "London", "UK"));**

**//var chris = john;**

**var chris = new Employee(john);**

**chris.Name = "Chris";**

**WriteLine(john); // oops, john is called chris**

**WriteLine(chris);}}}**

**Lesson04 Explicit Copy Constructor**

**Notes:-**

**1-with explicit copy constructor we define interface that receive generic type and with define method with return generic type and used inside every class and sub class as below (But this way is better that previous ways but its hard call coded).**

**//this interface define method called DeepCopy that implement on each class**

**namespace ExplicitDeepCopyPro.Interfaces{**

**public interface IProtoType<T>{T DeepCopy();}}**

**using ExplicitDeepCopyPro.Interfaces;**

**using System;**

**namespace ExplicitDeepCopyPro.Models{**

**public class Address : IProtoType<Address>{**

**public string StreetAddress, City, Country;**

**public Address(string streetAddress, string city, string country){**

**StreetAddress = streetAddress ?? throw new ArgumentNullException(paramName: nameof(streetAddress));**

**City = city ?? throw new ArgumentNullException(paramName: nameof(city));**

**Country = country ?? throw new ArgumentNullException(paramName: nameof(country));}**

**//to apply deep copy we pass the instnace and assign value with apply initialize of sub class**

**public Address(Address other){**

**StreetAddress = other.StreetAddress;**

**City = other.City;**

**Country = other.Country;}**

**public override string ToString(){**

**return $"{nameof(StreetAddress)}: {StreetAddress}, {nameof(City)}: {City}, {nameof(Country)}: {Country}";}**

**public Address DeepCopy(){**

**return new Address(StreetAddress, City, Country);}}}**

**using ExplicitDeepCopyPro.Interfaces;**

**using System;**

**namespace ExplicitDeepCopyPro.Models{**

**public class Employee : IProtoType<Employee>{**

**public string Name;**

**public Address Address;**

**public Employee(string name, Address address){**

**Name = name ?? throw new ArgumentNullException(paramName: nameof(name));**

**Address = address ?? throw new ArgumentNullException(paramName: nameof(address));}**

**//deep copy we pass the instnace and assign value with apply initialize of sub class**

**public Employee(Employee other){**

**Name = other.Name;**

**Address = new Address(other.Address);}**

**public override string ToString(){**

**return $"{nameof(Name)}: {Name}, {nameof(Address)}: {Address}";}**

**//we see that we call DeppCopy() of sub class inside the base class as below**

**public Employee DeepCopy(){**

**return new Employee(Name, Address.DeepCopy());}}}**

**//on the main entry we apply the explicit copy as below**

**static void Main(string[] args){**

**var john = new Employee("John", new Address("123 London Road", "London", "UK"));**

**var chris = new Employee(john);**

**chris.Name = "Chris";**

**WriteLine(john); // oops, john is called chris**

**WriteLine(chris);}**