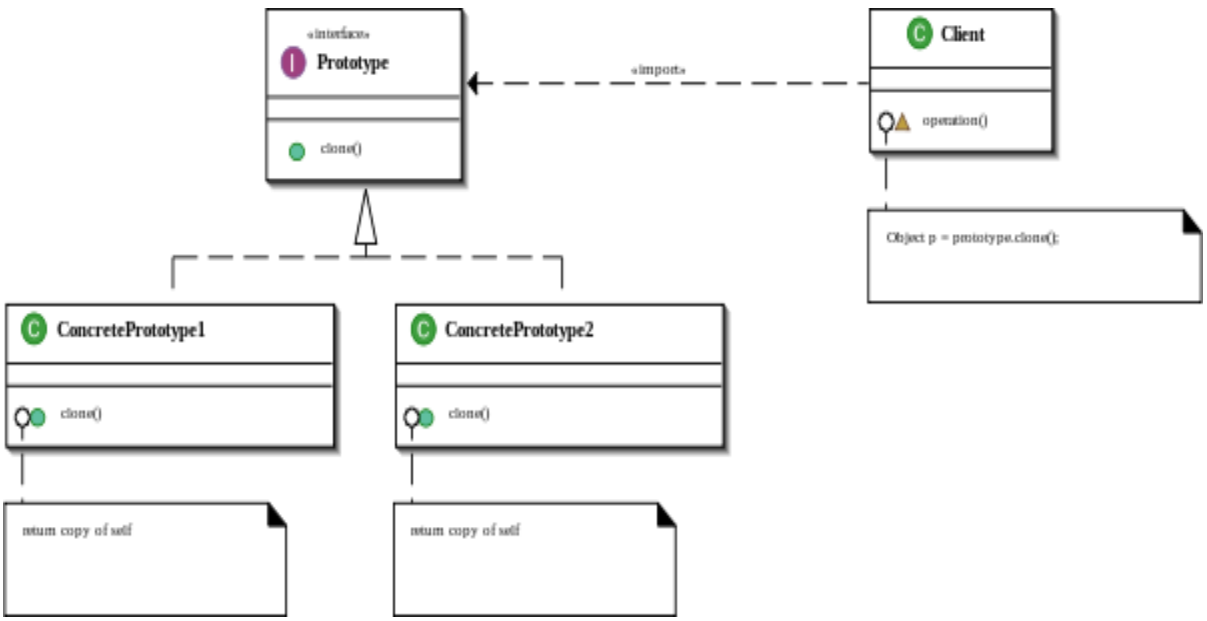
Prototype Pattern Implementation Lab Task 1

Introduction & Concept

In this lab task we will learn how to implement the prototype pattern using C#.Net

- Prototype pattern is a creational design pattern that lets you copy existing objects without making your code dependent on their classes.
- This pattern provides an alternative method for instantiating new objects by copying or cloning an instance of an existing object. You can avoid the expense of creating a new instance using this concept.
- It avoids subclasses of an object creator in the client application, like the factory method pattern does.
- It avoids the inherent cost of creating a new object in the standard way (e.g., using the 'new' keyword) when it is prohibitively expensive for a given application.
- The following is the structure of prototype pattern.




Aamir Shabbir Pare

Implementation

1. Create a console application in visual studio.
2. Create a folder and name it PrototypeImplementation
3. Create ***ICloneable.CS*** file in this folder and write the following code in it:

```
/// <summary>
/// ProtoType Abstract Class
/// </summary>
public abstract class ProtoType
{
    public abstract ProtoType Clone();
}
```

The above created class is an abstract class declaring a single method Clone(). It will be used as a prototype for all other subclass that we are going to create next in this lab task.

In the next step we are going to create concreate classes that will inherit from the ProtoType abstract base class created above.

4. Create **ConcretePrototype1.cs** file in this folder and write the following code in it:

```
/// <summary>
/// ConcretePrototype1 class inherits from the
    ProtoType base class
/// </summary>
public class ConcretePrototype1 : ProtoType
{
    public int Id { get; set; }
    public string Name { get; set; }
    public override ProtoType Clone()
    {
        return (ProtoType)this.MemberwiseClone();
    }
}
```

In the above class we inherited from the ProtoType abstract base class and provide implementation to the Clone method. The MemberWieseClone() is used to create a copy of the object instead of using new keyword to create an object from scratch. MemberwiseClone() is a .Net Framework method which creates a shallow copy of the object.

Next, we are going to repeat process for step 5.

5. Create **ConcretePrototype2.cs** file in this folder and write the following code in it:

```
/// <summary>
/// ConcretePrototype2 class inherits from the
    ProtoType base class
/// </summary>
public class ConcretePrototype2 : ProtoType
{
    public int Id { get; set; }
    public string Name { get; set; }
    public override ProtoType Clone()
    {
        return (ProtoType)this.MemberwiseClone();
    }
}
```

The above code is as similar as in the step 4, the only difference is a class name. Here, the point in repeating the same process is to make you clear that the implementation of prototype will remain same for classes which inherits from base ProtoType.

Now we have implemented the ProtoType Pattern basic requirements and wrote the behaviors for clone method so that we are able to create clone copies rather than creating concrete objects from scratch.

Next, we are going to create a client class that is responsible of creating clones of already existing prototype instances.

6. Create ***Client.cs*** file in this folder and write the following code in it:

```
/// <summary>
/// Client that uses the ProtoType to create clones
/// rather than creating a new object using new keyword
/// </summary>
public class Client
{
    public static void Main(string[] args)
    {
        //Original copy of the ConcreteProtoType1
        //Used as a Prototype
        ProtoType original = new ConcreteProtoType1
        {
              Id = 10,
              Name = "Name of the ConcretePrototype1"
        };

        //Cloned Copy of the ConcreteProtoType1
          using Prototype
        var prototype = original.Clone();

        //Original copy of the ConcreteProtoType2
        //Used as a Prototype
        original = new ConcreteProtoType2
        {
              Id = 11,
              Name = "Name of the ConcretePrototype2"
        };

        //Cloned Copy of the ConcreteProtoType2
          using Prototype
        prototype = original.Clone();

        Console.ReadKey();
    }

}

}
```

The above client class creates original objects for each concrete class, used as prototype for creating clones of those already exiting prototypes. Now rather than using a new keyword to create objects from scratch that is expensive, we can use prototype to create copies of the object instance.

==========END TASK==========

Prototype Pattern Implementation Lab Task 2

In the following code make changes to the cloned copies and write the results on console. Display object states in the following manner:

- Display original copy state.
- Change cloned copy state.
- Display cloned copy state.
- Display original copy state.

Now observe the object states.

```
1.  /// <summary>
    /// Client that uses the ProtoType to create clones
    /// rather than creating a new object using new keyword
    /// </summary>
    public class Client
    {
        public static void Main(string[] args)
        {
            //Original copy of the ConcreteProtoType1
             //Used as a Prototype
            ProtoType original = new ConcreteProtoType1
             {
                    Id = 10,
                    Name = "Name of the ConcretePrototype1"
             };

            //Cloned Copy of the ConcreteProtoType1
               using Prototype
            var prototype = original.Clone();

            //Original copy of the ConcreteProtoType2
            //Used as a Prototype
            original = new ConcreteProtoType2
             {
                    Id = 11,
                    Name = "Name of the ConcretePrototype2"
             };

            //Cloned Copy of the ConcreteProtoType2
               using Prototype
            prototype = original.Clone();
```

Aamir Shabbir Pare

```
            Console.ReadKey();
        }

    }

}
```

Write your observations that you found in this program by reflecting changes in cloned copies of original instance.

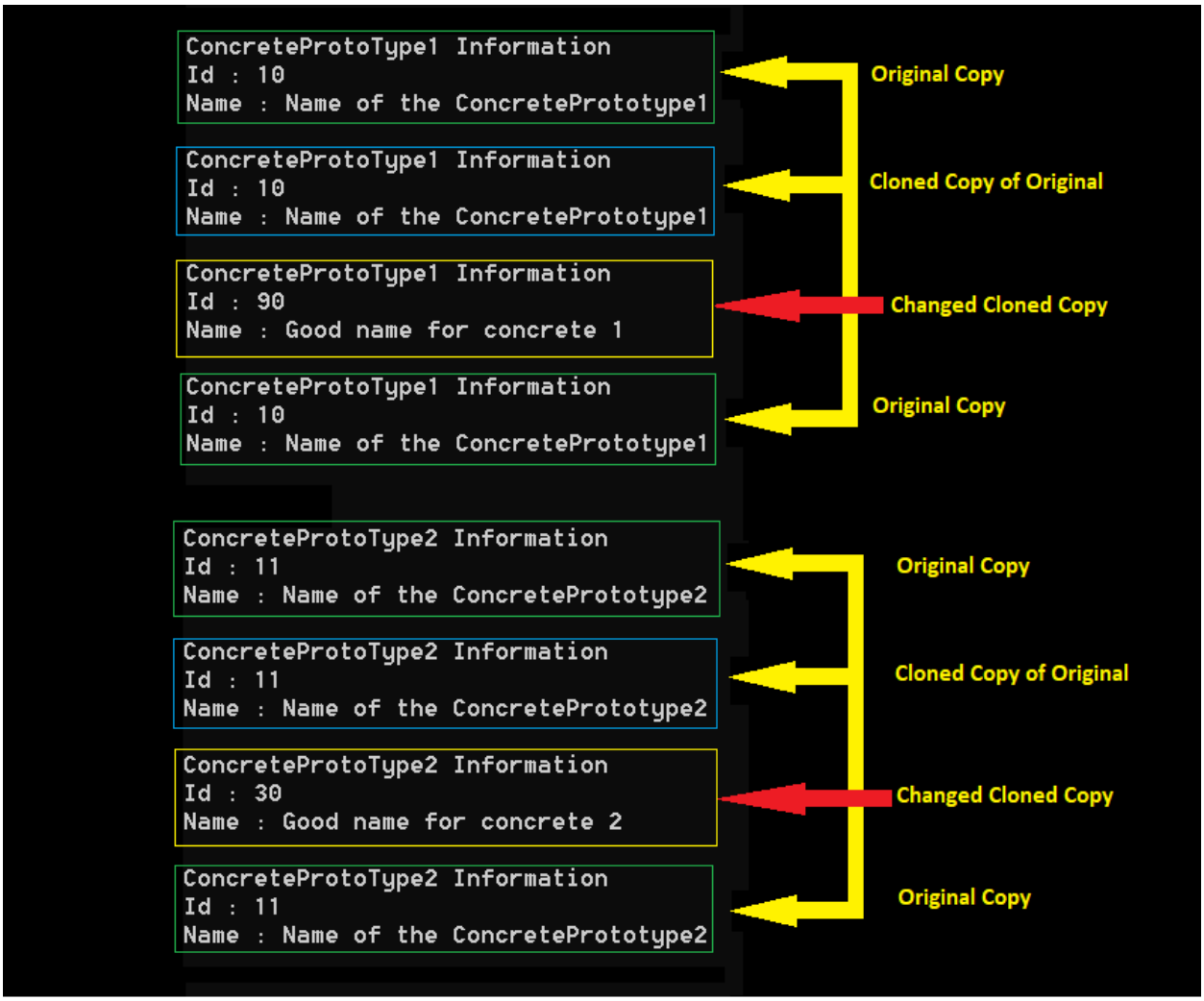
==========END TASK==========

Following code demonstrates the Lab Task 2

```
/// <summary>
/// Client that uses the ProtoType to create clones
/// rather than creating a new object using new keyword
/// </summary>
public class Client
{
    public static void Display(ProtoType p)
    {
        if (p.GetType() == typeof(ConcreteProtoType1))
        {
            var prototype = (ConcreteProtoType1)p;
            Console.WriteLine(
                $"{prototype.GetType().Name} Information");
            Console.WriteLine($"Id : {prototype.Id}");
            Console.WriteLine($"Name : {prototype.Name}");
        }

        if (p.GetType() == typeof(ConcreteProtoType2))
        {
            var prototype = (ConcreteProtoType2)p;
            Console.WriteLine(
                $"{prototype.GetType().Name} Information");
            Console.WriteLine($"Id : {prototype.Id}");
            Console.WriteLine($"Name : {prototype.Name}");
        }
        Console.WriteLine();

    }
```

```
        public static void Main(string[] args)
        {
            //Original copy of the ConcretePrototype1
            ProtoType prototype = new ConcretePrototype1
            {
              Id = 10,
              Name = "Name of the ConcretePrototype1"
            };
            Display(prototype);

            //Cloned Copy of the ConcretePrototype1 using Prototype
            var copy1 = (ConcretePrototype1)prototype.Clone();
            Display(copy1);

            //Changed in the copy
            copy1.Id = 90;
            copy1.Name = "Good name for concrete 1";
            Display(copy1);
            Display(prototype);
            Console.WriteLine();

            //Original copy of the ConcretePrototype2
            prototype = new ConcretePrototype2
            {
              Id = 11,
              Name = "Name of the ConcretePrototype2"
            };
            Display(prototype);

            //Cloned Copy of the ConcretePrototype2 using Prototype
            var copy2 = (ConcretePrototype2)prototype.Clone();
            Display(copy2);
            //Changed copy
            copy1.Id = 30;
            copy1.Name = "Good name for concrete 2";

            Display(copy2);
            Display(prototype);
            Console.ReadKey();
        }
    }
}
```

The Output after changing cloned Copies of each instance.



Prototype Pattern Implementation Lab Task 3

Implement Prototype on the following student-course scenario. The student with other its linear properties declares a complex course property. With shallow copy like MemberwiseCopy() it's not possible to get a cloned copy of it. This scenario requires the deep copy concept to solve the problem.

Use following classes for this task.

```
public class Course
{
    public int CourseId { get; set; }
    public string Title { get; set;
}
public class Student
{
    public Course Course { get; set; }
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Goodbye, wish you all the best and see you in next lab task!