FLYWEIGHT PATTERN

1/Problem: Game Alien Hunting

We are writing a game for a Smartphone where the amount of memory is very limited and you need to show many aliens that are identical in shape, you can have only one place that holds the shape of the alien instead of keeping each identical shape in the precious memory.

2/ Solution with normal way:

Creat an array of class alien. In this class, there are all variables

```
public enum Color { Green, Red, Blue };

public class alien
{
	string shape;
	color aliencolor;
};
```

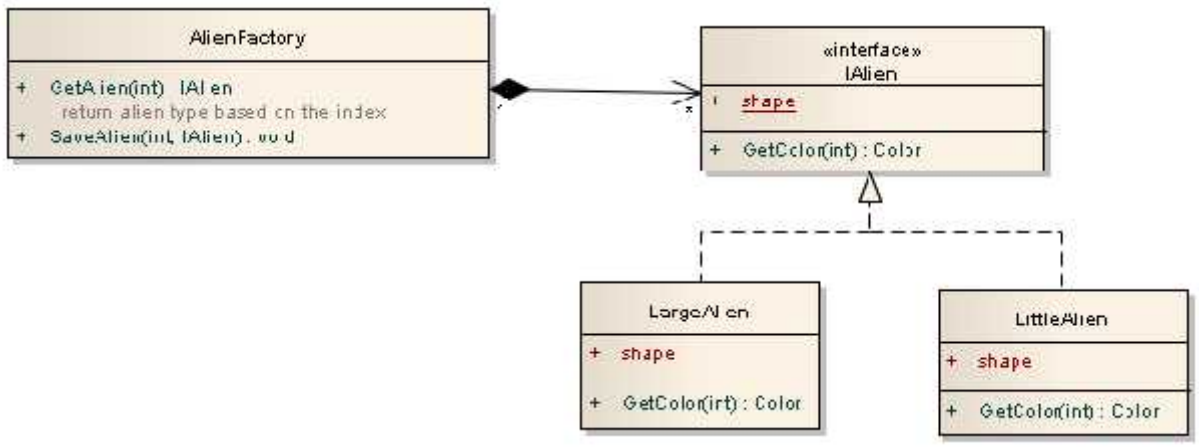
Then just input one by one

3/ Problem with normal solution:

If I do this way, with aliens have the same shape, I always creat variable shape to contain value, which means it waste a lot of memories to store information. What if the data is not only shape ? Obviously, you will loose lots of memories

4/ Solution with flyweight pattern:

The flyweight pattern is used to allow the application to point to the same instance of the object to save memory (hence it's a structural pattern). The shapes of the aliens are all the same, but their color will change based on how mad each are. The shapes of the aliens will be Intrinsic, and the color of the alien will be Extrinsic.



- The AlienFactory class stores and retrieves different types of aliens usingits GetAlien and SaveAlien methods .
- The IAlien interface defines the shape property as the intrinsic state and the GetColor method as the extrinsic state.
- The LargeAlien and the LittleAlien classes are the flyweight objects where each has its own shapeintrinsic state and ways of calculating the GetColor extrinsic state.

Source code:

```
public enum Color { Green, Red, Blue }

class Program
{
    static void Main(string[] args)
    {
        //create Aliens and store in factory
        AlienFactory factory = new AlienFactory();
        factory.SaveAlien(0, new LargeAlien());
        factory.SaveAlien(1, new LittleAlien());

        //now access the flyweight objects
        IAlien a = factory.GetAlien(0);
        IAlien b = factory.GetAlien(1);

        //show intrinsic states, all accessed in memory without calculations
        Console.WriteLine("Showing intrinsic states...");
        Console.WriteLine("Alien of type 0 is " + a.Shape);
        Console.WriteLine("Alien of type 1 is " + b.Shape);

        //show extrinsic states, need calculations
        Console.WriteLine("Showing extrinsic states...");
        Console.WriteLine("Alien of type 0 is " + a.GetColor(0).ToString());
        Console.WriteLine("Alien of type 0 is " + a.GetColor(1).ToString());
        Console.WriteLine("Alien of type 1 is " + b.GetColor(0).ToString());
        Console.WriteLine("Alien of type 1 is " + b.GetColor(1).ToString());
    }
}

public interface IAlien
{
    string Shape { get; }  //intrinsic state

    Color GetColor(int madLevel);  //extrinsic state
}

public class LargeAlien : IAlien
{
    private string shape = "Large Shape";  //intrinsic state

    string IAlien.Shape
    {
        get { return shape; }
    }

    Color IAlien.GetColor(int madLevel)   //extrinsic state
    {
        if (madLevel == 0)
            return Color.Green;
```

```
        else if (madLevel == 1)
            return Color.Red;
        else
            return Color.Blue;
    }
}

public class LittleAlien : IAlien
{
    private string shape = "Little Shape";  //intrinsic state

    string IAlien.Shape
    {
        get { return shape; }
    }

    Color IAlien.GetColor(int madLevel)   //extrinsic state
    {
        if (madLevel == 0)
            return Color.Red;
        else if (madLevel == 1)
            return Color.Blue;
        else
            return Color.Green;
    }
}

public class AlienFactory
{
    private Dictionary<int,> list = new Dictionary<int,>();

    public void SaveAlien(int index, IAlien alien)
    {
        list.Add(index, alien);
    }

    public IAlien GetAlien(int index)
    {
        return list[index];
    }
}</int,>
```

5/ Note:

Flyweight pattern is not a “has-a” relationship, but it is realization. When you implement, all elements must have the same class type (based class). Derived classes just help you classify elements with the same characteristics or datas to save memories.

6/ Similar problems:

- The Flyweight uses sharing to support large numbers of objects efficiently. Modern web browsers use this technique to prevent loading same images twice. When browser loads a web page, it traverse through all images on that page. Browser loads all new images from Internet and places them the internal cache. For already loaded images, a flyweight object is created, which has some unique data like position within the page, but everything else is referenced to the cached one.

- The war game instantiates 5 Soldier clients, each client maintains its internal state which is extrinsic to the soldier flyweight. And Although 5 clients have been instantiated only one flyweight Soldier has been used

6/ Advantages and Disadvantages of Flyweight Pattern

- Advantages:

- It reduces the number of objects created.
- It reduce memory requirements and instantiation time and related costs.
- One object of a class can provide many virtual instances.

- Disadvantages:

- If poorly managed, it can significantly complicate our application architecture, as it introduces many small but similar objects into our namespace.

- developers unfamiliar with the pattern may have a hard time grasping why it's being used.