# KF7012 Week 6 Lab– Implementing Object Orientated Designs

## Introduction

During the lecture this week we discussed several creational design patterns. These patterns are there as a guide to best practice, they show a template for a solution which can be adapted to the needs of the application. This week lab we will look at building a simple Object Pool. The pool will have a minimum size and a maximum size. And the Pool will be strongly typed using generics i.e. when creating the pool we will have to specify the type of object that it will contain. This adds a level of complication and makes the implementation of a simple singleton pattern a technical challenge so we will not be implementing the Pool with the Singleton which would be normal practice.

The Pool requires no only its type at the time of creation but also a template object which can be cloned; this template object will be of the same type as the pools declared type (generics). This is similar to when we have used list<T> when we have to declare the type.

Optionally the pool will also allow it’s Maximum and Minimum size to be specified at the time of creation. The default size of none is specified is minimum = 2, maximum = 3.

So why would we ever want a pool, the simple answer is that of performance. Object instantiation is a very slow process if we are able to reuse a objects which have already been created by simply changing their values via setter methods, then there should be a performance improvement. Obviously with any pattern you have to have a specific need before you would implement.

Constructors

public ObjectPool(int minSize, int maxSize, T objectToClone)  
public ObjectPool(T objectToClone)

If we look at the class diagram below we can see that there is a maxSize, minSize, and O

objectToClone these are the three elements which can be specified on the constructor. Pool is the collection of objects that we are managing, An array list is similar to a list except that it is not a strongly typed collection, and therefore not part of the generics collection you will need to use

using System.Collections; There was a decision to use an arraylist after problems with using the collection which was strongly types having the same T (template) variable seemed cause problems for C# .

Note that this class needs to use Generics the <T> on the class name is then used to check that the elements added are of the correct type (type T)

public class ObjectPool<T>



There should be two constructors to the class, both are parameterised. The first should take the minimum size, maximum size and an Object of the type which the Pool is to manage; this is a template object which will be used to create other instances. The second constructor will take only the template object and sets the minimum size to 1 and maximum size to 2.

public ObjectPool(int minSize, int maxSize, T objectToClone)  
public ObjectPool(T objectToClone)

The constructor will use the method AddClones to initially set the size of the list to the minimum size.

private void AddClones(int numberToAdd)  
This method adds a number of clones of the template object to the ArrayList, it should be called by the constructor to add the initial minimum set of objects to the Pool.

private static T DeepClone<T>(T obj)

This is essentially the DeepCopy you have already seen during your lectures, note that any object which you want to deep copy using the serialization process will have to have been marked as serilizable.

You will need to look back in your notes to see the method for the deepcopy process. You will need to add the following libraries to get the code working.

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

public T CheckOut()

This method will return an object of T to the client; the object will need to be removed from the pool. Additionally you should check that the list size will not fall below the minimum size, if the number of objects in the pool is less than the minimum then you need use AddClones add additional objects.

public void CheckIn(T obj)

This is more complicated than just adding the object back into the pool, firstly you should check that the reference you are being passed is not a reference of one of the existing objects in the pool. We need to do this as we need to guarantee that every element in the pool is unique, don’t trust other programmers! You can use the methods Object.ReferenceEquals(Obj1,Obj2) Which checks for pointer equality.

After adding elements back into the pool you need to check its size, if larger than the maximum after the checkin you will need to adjust the size by removing elements.

Test you program is working as expected you can do this by creating an the ObjectPool with StringBuilder objects and make sure that it functions correctly, i.e. stops the same object from being entered into the pool twice, increases size when needed, decreases size when needed. You may wish to add a method to the pool to aid in this process, one that will return the count of the objects in the pool.

ThreadSafty, a pool should be thread safe, below is a reference to a simple article on locking the threads.

See <http://msdn.microsoft.com/en-us/library/c5kehkcz.aspx> add thread locking to the application.

The pool should be a singleton. However this is outside the scope of today’s work you can look at in your own time if you have the time needed to research the different possibilities, the two links below should be a starting point there are other possibilities such as reflections.

Source: <http://stackoverflow.com/questions/4203634/singleton-with-parameters>

<http://stackoverflow.com/questions/2319075/generic-singletont>