Using Events and Delegates

In this demo we will implement an event delegate concepts. Events are special types of delegates used for notification.

Step 1: Open VS2010 and create a new project. Name this project as 'CS_Event_Delegate'.

Step 2: In the same project add a new class file, name it as 'DataFiles.cs'. Here you will get class DataFiles creared, remove the class and write the following code inside 'CS_Event_Delegate' namespace:

```
//Declaring Delegate
    public delegate void TransactionCahnged(decimal trAmt);

    public class clsBanking
    {
        public int AccNo { get; set; }
        public string AccName { get; set; }
        public decimal OpeningBalance { get; set; }
        public decimal TrAmount { get; set; }
        public decimal NetBalance { get; set; }
    }

    public class BankingBizAction
    {
        clsBanking _objBank;


        //Declaring Event
        public event TransactionCahnged OverBalance;

        //public event <DelegateName> <EventNAme>

        public event TransactionCahnged UnderBalance;

        public void CreateAccount(clsBanking objBank)
        {
            _objBank = objBank;
        }

        public void Deposit(decimal trAmt)
        {
            _objBank.NetBalance = _objBank.OpeningBalance +
trAmt;

            if (_objBank.NetBalance >= 100000)
            {
```

```
                //Raising Event
                OverBalance(trAmt);
                //<event Name>(parameter)
            }
        }

        public void Withdraw(decimal trAmt)
        {
            _objBank.NetBalance = _objBank.OpeningBalance -
trAmt;

            if (_objBank.NetBalance <= 5000)
            {
                //Raising Event
                UnderBalance(trAmt);
            }
        }

        public decimal GetNetBalance()
        {
            return _objBank.NetBalance;
        }
    }

    public class EventListener
    {
        BankingBizAction _objBiz;

        public EventListener(BankingBizAction bt)
        {
            _objBiz = bt;

            //Subscribing and Handling Event
            //<ActionCloassObject>.<EventName> += new
<Delegate>(<Method to be used for notifications>)
            _objBiz.OverBalance += new
TransactionCahnged(_objBiz_OverBalance);
            _objBiz.UnderBalance += new
TransactionCahnged(_objBiz_UnderBalance);
        }

        void _objBiz_UnderBalance(decimal trAmt)
        {
            Console.WriteLine("Your current balance is Rs." +
_objBiz.GetNetBalance());
            Console.WriteLine("You need to maintain Rs 5000 in
account");
```

```
        }

        void _objBiz_OverBalance(decimal trAmt)
        {
            decimal overBalance = _objBiz.GetNetBalance() -
100000;
            decimal Tax = overBalance * Convert.ToDecimal(0.2);
            Console.WriteLine("Your Balance is Rs." +
overBalance + "/- more than Rs. 100000");
            Console.WriteLine("So Please Pay Tax of Rs." + Tax +
"/- Immediately");
        }
    }
```

The above code declares delegate 'TransactionChanged', this will be used to declare events that's whay the return type of the delegate is 'void'. The class 'clsBanking' is used to store Bank caustomer details. The class BankingBizAction, declares methods like CreateAccount(), Deposit() and Withdrawal(). The class also declared events 'OverBalance' and 'UnderBalance' using delegate declared at namespace level. In Deposit() and Withdrawal() methods these events are raised.

Since Events raised needs to be listened, the above code also contains 'EventListener' class. The constructor accepts the object of 'BankingBizAction' class to notify to events. The constructor of the 'EventListener' class subscribes and Handles events using delegate (+=) syntax.

Step 3: Open Program.cs and write the following code in Main method:

```
clsBanking objBank = new clsBanking();


            objBank.AccNo = 1001;
            objBank.AccName = "Mahesh Sabnis";

            objBank.OpeningBalance = 50000;

            objBank.TrAmount = 48000;

            BankingBizAction objBankBiz = new BankingBizAction();

            EventListener evt = new EventListener(objBankBiz);


            objBankBiz.CreateAccount(objBank);

            objBankBiz.Withdraw(objBank.TrAmount);

            Console.WriteLine("Net Balance = " +
```

```
objBankBiz.GetNetBalance() );


            Console.ReadLine();
```

Step 4: Run the application, if the condition for events matched then event will be raised and the result will ne displayed.

Generics

Generics are new in .NET 2.0 Framework. It is a mechanism of type safe data definition. In .NET 1.x we were having boxing and unBoxing. Typically UnBoxing utilize extra CPU cycles and hence reduce performance of the application. In this lab we will see how to declare a generic class. The class will provides facility of data structure. Generic uses "T" parameter, stands for Template. The generic class can be instantiated to specific .NET CLR of user defined type. Based upon this type a copy of the generic for that type in binary form is created in memory and used directly from the memory. This improves performance of the application.

Step 1: Open VS2010 and create a new project. Name it as 'Generic'. In this project add a new class, name it as 'GenericStack'.

Step 2: Write the following code in the class:

```
public class GenericStack<T>
    {
        T[] arr = new T[10];
        private int top = 0;

        public void Push(T itm)
        {
            arr[top++] = itm;
        }
        public T Pop()
        {
            return (arr[--top]);
        }
        public T [] Display()
        {
            return arr;
        }
    }
```

Step 3: Write the following code in Main() method:

```
//Stack object for Integer
            GenericStack<int> stkInt = new GenericStack<int>();
            stkInt.Push(10);
            stkInt.Push(20);
            stkInt.Push(30);
            stkInt.Push(40);
            stkInt.Push(50);
            stkInt.Push(60);
            stkInt.Push(70);
            stkInt.Push(80);
            stkInt.Push(90);
            stkInt.Push(100);

            foreach (int item in stkInt.Display())
            {
                Console.WriteLine("Item \t" + item);
            }

            //Stack object for Employees

            GenericStack<clsEmployee> stkEmp = new
GenericStack<clsEmployee>();

            stkEmp.Push(new clsEmployee() {EmpNo  = 1001, EmpName =
"Emp_1", DeptName = "Dep_1", Salary = 34001 });
            stkEmp.Push(new clsEmployee() { EmpNo = 1002, EmpName =
"Emp_2", DeptName = "Dep_2", Salary = 34002 });
            stkEmp.Push(new clsEmployee() { EmpNo = 1003, EmpName =
"Emp_3", DeptName = "Dep_3", Salary = 34003 });
            stkEmp.Push(new clsEmployee() { EmpNo = 1004, EmpName =
"Emp_4", DeptName = "Dep_4", Salary = 34004 });
            stkEmp.Push(new clsEmployee() { EmpNo = 1005, EmpName =
"Emp_5", DeptName = "Dep_5", Salary = 34005 });
            stkEmp.Push(new clsEmployee() { EmpNo = 1006, EmpName =
"Emp_6", DeptName = "Dep_6", Salary = 34006 });
            stkEmp.Push(new clsEmployee() { EmpNo = 1007, EmpName =
"Emp_7", DeptName = "Dep_7", Salary = 34007 });
            stkEmp.Push(new clsEmployee() { EmpNo = 1008, EmpName =
"Emp_8", DeptName = "Dep_8", Salary = 34008 });
            stkEmp.Push(new clsEmployee() { EmpNo = 1009, EmpName =
"Emp_9", DeptName = "Dep_9", Salary = 34009 });
            stkEmp.Push(new clsEmployee() { EmpNo = 1010, EmpName =
"Emp_10", DeptName = "Dep_10", Salary = 34010 });
```

```
            Console.WriteLine("EmpNo\t\tEmpName\t\tDeptName\tSalary");
            foreach (clsEmployee Emp in stkEmp.Display())
            {
                Console.WriteLine(Emp.EmpNo + "\t\t" + Emp.EmpName +
"\t\t" + Emp.DeptName + "\t\t" + Emp.Salary);
            }
            Console.ReadLine();
```

The above code defines instances GenericStack class for interger and clsEMployee type. Run the application, you will get all integers and Employees listed.

Self Study:

Try following classes

List<T>, SortedList<T>

Extension Methods

Extension Methods is the facility provided for adding logical group methods in the class which is sealed or cannot be extended. Following are rules:

- The class supposed to be container for extension method must be static.
- The method supposed to be act as extension method must be static.
- The first parameter of this method must be the reference of the class in which this method is added as an extension method.

Step 1: Open VS2010 and create a new project. Name this as 'CS_ExtensionMethod'.

Step 2: In this project add a new class as below:

```
public sealed class clsMath
    {
        public int Add(int x, int y)
        {
            return x + y;
        }

        public int Sub(int x, int y)
        {
            return x - y;
        }
```

```
    }
```

If you carefully reads this class, this is declared as sealed, means it can be extended.

Step 3: In the same project add new class, name it as 'MyExtensionClass'. This class is supposed to act as the container for extension method. It satisfy all the rules given above. The code of the class is as below:

```
public static class MyExtensionClass
    {
        public static int Mul(this clsMath m1,int x,int y)
        {
            return x * y;
        }

        public static int StringLenght(this string s)
        {
            int len = 0;

            foreach (char item in s)
            {
                len++;
            }

            return len;
        }
    }
```

In the above class, Mul method is static and has first parameter is an reference of the 'clsMath' class. In this class, this method is added as an extension method. Similarly we are adding extension method in string class of name 'StringLenght'.

Step 4: Now call these methods in 'Main()' method as below:

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

            clsMath objM1  =new clsMath();

            Console.WriteLine("Add = " + objM1.Add(2,3) );

            Console.WriteLine("Sub = " + objM1.Sub(2, 3));

            Console.WriteLine("Mul =" + objM1.Mul(2,4));


            string s = "SEED Infotech Ltd. Pune";

            Console.WriteLine("Lenght of" + s + " is = " +
```

```
s.StringLenght() );

            Console.ReadLine();
        }
```

In the above code, you will experience, though 'Mul()' and StrinLength()' methods are not physically present in class 'clsMath' and 'String' classes respectively, but still they are accessible using object of the concern classes.

Using LambdaExpression

In C# 3.0 Lambda expression is an simplified syntax for Anonymous Methods provided in C# 2.0. Lambda Expression internally uses delegate and Boolean parameter to evaluate an expression. C# 3.0 extensively uses Lambda Expression in LINQ.

Step 1: Open Vs2010 and create a new project. Name it as 'CS_LambdaExpression'.

Step 2: Write the code below in Program class:

```
static void Main(string[] args)
        {
            Console.WriteLine("C# 1.x Syntax");

            MyDelegate d1 = new MyDelegate(Increament);
            DoWork(d1);
            Console.WriteLine("Ends Here");
            Console.WriteLine();

            Console.WriteLine("C# 2.0 Anonymous Method");

            MyDelegate d2 = delegate(int x)
            {
                return x + 10;
            };

            DoWork(d2);

            Console.WriteLine("Ends Here");
            Console.WriteLine();


            Console.WriteLine("C# 2.0 Simple Syntax");

            DoWork(delegate(int x){return x + 10;});
```

```
            Console.WriteLine("Ends Here");
            Console.WriteLine();

            Console.WriteLine("C# 3.0 Lambda Expression");

            DoWork(x => x + 10);

            Console.WriteLine("Ends Here");

            Console.ReadLine();
        }


        static int Increament(int x)
        {
            return x + 10;
        }

        static void DoWork(MyDelegate d)
        {
            Console.WriteLine(d(10));
        }
```

The above code explains, Delegate , using delegate in actual scenario.