Payoda – Phase 2
Day 2 – C# Training

Implementation of Multi Level Inheritance and Method Overriding – Run Time Polymorphism

```
namespace PayodaDay2
{
   internal class Employee //Base Class
   {
       int EmpId;
       string EmpName;

      public Employee(int EmpId,string EmpName)
      {
         this.EmpId = EmpId;
         this.EmpName = EmpName;
      }
      public virtual void Display()
      {
         Console.WriteLine("EmpId"+EmpId+" "+"EmplName:"+EmpName);
      }
   }

}

namespace PayodaDay2
{
   internal class Department : Employee //Derived class
   {
      public string DeptName;
      public int Salary;

      public Department(int id,string empname,string dname,int sal) :base(id,empname)
      {
         DeptName = dname;
         Salary = sal;
      }
      public override void Display()
      {
         base.Display();
         Console.WriteLine(DeptName + " " + Salary);
      }
   }
}
```

```
namespace PayodaDay2
{
   internal class Manager : Department //Manager - derived class, Department - Base class
   {
      public string ManagerName;

      public Manager(int id,string ename,string dname,int salary,string managerName)
:base(id,ename,dname,salary)
      {
         ManagerName = managerName;
      }

      public override void Display()
      {
         base.Display();
         Console.WriteLine(ManagerName);
      }
   }
}

namespace PayodaDay2
{
   internal class Program
   {
      private static void Main(string[] args)
      {
         //Object Initializer - At the time of instance creation,class members can be initialized
         //Employee employee = new Employee() { EmpId = 11, EmpName = "Reks" };
         //employee.Display();

         Manager mgr = new Manager(11, "Anu", "HR", 90000, "Ram");
         mgr.Display();
         //dpt.Show();
      }
   }
}
```

Method Overloading – Compile Time Polymorphism

```
internal class Program
{
   //Method Overloading - Same method but different parameters 1.Type of parameters 2.Number of
Parameters 3.Return type
   public void Add(int a,int b)
   {
      Console.WriteLine(a + b);
   }

   public void Add(int a,int b,int c)
   {

   Console.WriteLine( a+b+c);
   }

   public double Add(double a, double b)
   {
      return a + b;
   }
   public void Add(string a, string b)
   {
      Console.WriteLine(a + b);
   }
   private static void Main(string[] args)
   {
      Program pgm = new Program();
      pgm.Add("Priya", "Mohan");
      pgm.Add(123, 34, 20);
   }
}
```

Implementation of get and set method, Abstract class

```
namespace gettersettersamp
{
   class Product
   {
      int ProId;
      string ProName;
      int ProPrice;

      //treat the private as public
      public int Id
      {
         get
         {
            return ProId;
         }
         set
         {
            ProId = value;
         }
      }
      public string Name
      {
         get
         {
            return ProName;
         }
         set
         {
            ProName = value;
         }
      }
      public int Price
      {
         get
         {
            return ProPrice;
         }
         set
         {
            if(value>60000)
            {
               ProPrice = value;
            }
            else
            {
               Console.WriteLine("The Price should be greater than 50000" );
            }
         }
      }
```

```
     public Product()
      {

      }
      public Product(int id,string name,int pri)
      {
         ProId=id;
         ProName=name;
         ProPrice=pri;
      }
      public void Display()
      {
         Console.WriteLine(Id + " " + Name + " " + Price);
      }
   }
   internal class Program
   {
      private static void Main(string[] args)
      {
         Product pro = new Product(111,"Mac",999);
         pro.Display();
         Product pro1 = new Product() {Id=112,Name="Laptop",Price=6000};
         //creating instance to customersalary by referencing to abstract class customer
         Customer cu = new CustomerSalary() { CustId=11,Name="Anu",Sal=60000};
         cu.BasicInfo();
         cu.SalaryInfo();
       }
   }
}
namespace gettersettersamp
{
   abstract class Customer
   {
      public int CustId { get; set; } //autoimplemented property
      public string Name { get; set; }

      public void BasicInfo()
      {
         Console.WriteLine(CustId + " " + Name);
      }
     public abstract void SalaryInfo();
   }
   class CustomerSalary : Customer
   {
      public int Sal { get; set; }
      public override void SalaryInfo()
      {
         Console.WriteLine($"Salary: {Sal}");
      }
   }
}
```

Abstract Class

```
namespace Abstractdemo
{
   abstract class Flight
   {
      public int FlightNo { get; set; }
      public string FlightName { get; set; }

      public void FlightDetails()
      {
         Console.WriteLine($"FlightNo:{FlightNo}, FlightName:{FlightName}");
      }
      public abstract void FareDetails();
   }
}

namespace Abstractdemo
{
   internal class FlightFare : Flight
   {
      public decimal BasicFare { get; private set; }
      public int durationinhrs { get; set; }
      public string typedestination { get; set; }

      public override void FareDetails()
      {
         if (durationinhrs > 5)
         {
            BasicFare = 9000;
         }
         else
         {
            BasicFare = 5000;
         }
         if (typedestination.ToLower().Equals("international"))
         {
            BasicFare += 6000;
         }
         Console.WriteLine($"BasicFare:{BasicFare}");
      }
   }
}

internal class Program
{
   private static void Main(string[] args)
   {
      Flight ft = new FlightFare() { FlightNo = 11, FlightName =
"AirIndia",durationinhrs=6,typedestination="International"};
      ft.FlightDetails();
      ft.FareDetails();
```

```
    }
}
```

Interface Implementation

```
namespace InterfaceDemo
{
    internal interface IShape //public abstract
    {
        void CalculateArea();

        //Default Implementation is allowed after C#12
        void DefaultShape(int side)
        {
            Console.WriteLine(Math.Pow(side, 2));
        }
    }
    class Circle : IShape
    {
        public int radius { get; set; }
        public void CalculateArea()
        {
            Console.WriteLine(Math.PI * Math.Pow(radius, 2));
        }
    }
    class Rect : IShape
    {
        public int lnth { get; set; }
        public int bredth { get; set; }

        public void CalculateArea()
        {
            Console.WriteLine(lnth * bredth);
        }
    }
}

internal class Program
{
    private static void Main(string[] args)
    {
        Console.WriteLine("Enter the Shape you want to calculateArea: 1.Circle 2. Rectangle
3.DefaultShape");
        int ch = Convert.ToInt32( Console.ReadLine());

        if(ch==1)
        {
            Console.WriteLine("Enter the radius:");
            int r = Convert.ToInt32( Console.ReadLine());
            Circle cir = new Circle() { radius = r };
            cir.CalculateArea();
        }
        else if(ch==2)
```

```
        {
            Rect rct = new Rect() { lnth = 10, bredth = 20 };
            rct.CalculateArea();
        }
        else if(ch==3)
        {
            IShape sh = new Circle();
            sh.DefaultShape(5);
        }
        else
        {
            Console.WriteLine("Enter valid choice");
        }
    }
}
```