1) Create a hierarchy of Employee, Manager, MarketingExecutive in Employee Management System. They should have the following functionality.

a) Manager with following private members.

q Petrol Allowance: 8 % of Salary.

q Food Allowance : 13 % of Salary.

q Other Allowances : 3% of Salary.

Calculate GrossSalary by adding above allowances. Override CalculateSalary() method to calculate Net Salary. NetSalary. PF calculation should not consider above allowances.

b) MarketingExecutive with following private members.

q Kilometer travel

q Tour Allowances : Rs 5/- per Kilometer (Automatically generated).

q Telephone Allowances : Rs.1000/-

Calculate GrossSalary by adding above allowances. Override CalculateSalary(). NetSalary,PF calculation should not consider above allowances.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace employee1
{
   internal class Program
   {
   interface IPrintable
      {
         void Mprint();

      }
      public class Employee
      {
         public static void Main()
         {
            Console.Write("Enter the salary of an Employee to find Food ,Petrol,Other Allowances :");
            Manager m = new Manager(Convert.ToDouble(Console.ReadLine()));
            m.Foodie();
            m.Petrol();
            m.Others();
```

```
            m.GrossSalary();
            m.CalculateSalary();
            m.Mprint();

            Console.Write("Enter the salary of an Employee to find Tele and Tour Allowances:");

            MarketingExecutive me = new
MarketingExecutive(Convert.ToDouble(Console.ReadLine()));
            me.Grosssalary();
            me.CalculateSalary();
            me.Mprint();
        }

    }

  public class Manager : IPrintable
    {
        private double _Petrol;
        private double _Food;
        private double _Others;
        private double sal;
        private double Gross;
        private double Netsal;
        private double Pf, TDS;

        public Manager(double Esalary)
        {
            sal = Esalary;
        }
        public void Foodie()
        {
            _Food = sal * 0.13;
        }
        public void Petrol()
        {
            _Petrol = sal * 0.08;
        }
        public void Others()
        {
            _Others = sal * 0.03;
        }
        public void GrossSalary()
        {
            Gross = sal + _Food + _Petrol + _Others;
        }
        public void CalculateSalary()
        {
            Pf = (Gross * 0.10);
            TDS = (Gross * 0.18);
            Netsal = Gross - (Pf + TDS);
        }

        public void Mprint()
        {
            Console.WriteLine("Employee Petrol Allowances:{0}", _Petrol);
```

```
            Console.WriteLine("Employee Food Allowances:{0}", _Food);
            Console.WriteLine("Employee Petrol Allowances:{0}", _Others);
            Console.WriteLine("Employee Gross Salary with Allowances:{0}", Gross);
            Console.WriteLine("Employee Net Salary:{0}", Netsal);
        }

    }

  public class MarketingExecutive : IPrintable
    {
        private double sal;
        private double KM;
        private double TourAllowances;
        private double TelephoneAllowances;
        private double Netsal, Pf, TDS, Gross;
        public MarketingExecutive(double Esal)
        {
            this.sal = Esal;
        }
        public void Grosssalary()
        {
            Console.Write("Enter the Total Kilometers Covered:");
            KM = Convert.ToDouble(Console.ReadLine());
            TourAllowances = 5 * KM;
            TelephoneAllowances = 1000;
            Gross = sal + TourAllowances + TelephoneAllowances;
        }

        public void CalculateSalary()
        {
            Pf = (Gross * 0.10);
            TDS = (Gross * 0.18);
            Netsal = Gross - (Pf + TDS);
        }



        public void Mprint()
        {
            Console.WriteLine("Emp Travel Allowances:{0}", TourAllowances);
            Console.WriteLine("Emp Telephone Allowances:{0}", TelephoneAllowances);
            Console.WriteLine("Emp Gross salary with  Allowances:{0}", Gross);
            Console.WriteLine("Emp Net Salary :{0}", Netsal);
        }
    }
  }
}
```

2) Write a class called MyStack with following members.

a) integer array

b) integer variable to store top position

c) size of the array.

Implement Push() and Pop() operation. Implement ICloneable interface to perform cloning. Write a client application to perform cloning.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IcloneableDemo
{
interface Icloneable
{
void Push(int data);
int Pop();
}
class IcloneableDemo
{

public static void Main()

{
MyStack pp = new MyStack();

pp.Push(100);
pp.Push(25);
pp.Push(30);
pp.Push(40);
pp.Push(50);
pp.Push(60);
//pp.Push(70);
pp.print();
Console.WriteLine();
Console.WriteLine("Element removed from stack:" + pp.Pop());
Console.WriteLine("Element removed from stack:" + pp.Pop());

Console.WriteLine("Element removed from stack:" + pp.Pop());
Console.WriteLine("Element removed from stack:" + pp.Pop());
Console.WriteLine("Element removed from stack:" + pp.Pop());
Console.WriteLine("Element removed from stack:" + pp.Pop());
}
}

public class MyStack : Icloneable
{
```

```
int[] arr = new int[5];

int top = 0;



public void Push(int data)

{

try
{
if (top == 5)
{
Console.WriteLine("StackOverFlow");

}
else
{
arr[top] = data;
top++;
}

}
catch (Exception e)
{

}
}

public void print()
{

for (int i = top - 1; i >= 0; i--)
{
Console.WriteLine(arr[i]);
}
}
public int Pop()
{
if (top <= 0)
{
Console.WriteLine("Stack Under Flow");
return -1;
}
else
{
int temp = arr[top - 1];
top--;

return temp;
}
```

```
}
}
}
```

3) Create a custom exception class named StackException. The Push()and Pop() method should throw object of StackException when the stack is full or empty respectively.

StackException.cs

```
using CCA;
using System;

public class demo
{
static int MAX = 5;
int[] st = new int[MAX];
int top = -1;
bool isempty()
{

if (top == -1)
return true;
else
return false;
}

public void Push(int data)
{
try
{
if (top > MAX)
{
throw new StackException("Stack Overflow");

}
else
{
st[++top] = data;
top++;
}
}
catch(Exception e)
{
Console.WriteLine(e.Message);
}
}

public void Pop()
{
int data;
try
{
```

```
if (!isempty())
{
data = st[top];
top = top - 1;

}
else
{
throw new StackException("Stack Underflow");

}
}
catch(Exception e)
{
Console.WriteLine("Stack Underflow");
}

}

public static void Main(String[] args)
{
//Employee emp = new Employee();
//emp.EmployeeDetails();

int ch;
demo se = new demo();

do
{
Console.WriteLine("\n1.Push\n2.Pop\n3.Exit");
Console.WriteLine("\nEnter the operation");
ch = Convert.ToInt32(Console.ReadLine());
switch (ch)
{
case 1:
Console.WriteLine("Enter the value");
int data = Convert.ToInt32(Console.ReadLine());
se.Push(data);
break;
case 2:
se.Pop();
break;
case 3: break;
}
}
while (ch != 3);}}
```

program.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace CCA
{
public class StackException : Exception
{
public StackException() { }

public StackException(string message) : base(String.Format(message))
{

}
}
}
```