Assignment 3

## Assignments to be done in this session

1. Create a hierarchy of Employee, Manager, MarketingExecutive in Employee Management System. They should have the following functionality.
   1. Manager with following private members.
      * Petrol Allowance: 8 % of Salary.
      * Food Allowance : 13 % of Salary.
      * 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.

* 1. MarketingExecutive with following private members.
     + Kilometer travel
     + Tour Allowances : Rs 5/- per Kilometer (Automatically generated).
     + Telephone Allowances : Rs.1000/-

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

Implement IPrintable interface for every Employee which will allow to print details of Employee on console.

Program

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Console;

namespace EmployeeManagementSystem

{

public interface IPrintable

{

void DisplayEmployeeDetails();

}

public class Employee

{

public int emp\_id;

public string emp\_name;

public float Salary;

public float GrossSalary;

public void GetSalaryWithEmployeeDetails()

{

WriteLine("Enter Employee Details Like emp\_id,emp\_name");

emp\_id = int.Parse(ReadLine());

emp\_name = ReadLine();

WriteLine("Enter Salary of an Employe .");

Salary = float.Parse(ReadLine());

}

public virtual void CalculateSalary()

{

}

}

public class Manager : Employee,IPrintable

{

private float PA;

private float FA;

private float OA;

public override void CalculateSalary()

{

PA = 8 \* Salary / 100;

FA=13\*Salary / 100;

OA=3\*Salary / 100;

GrossSalary = Salary + PA + FA + OA;

}

public void DisplayEmployeeDetails()

{

WriteLine("Manager Details...");

WriteLine("emp\_id: {0}\nemp\_name: {1}\nGrossSalary:{2}",emp\_id,emp\_name,GrossSalary);

}

}

public class MarketingExecutive : Employee,IPrintable

//inherited Employee class and interface IEmployee

{

private int Distance;

private float TourAllowances;

private float Telephone\_Allowances;

public override void CalculateSalary() //override method of Employee class

{

WriteLine("Enter Distance travelled during Marketing Executive in km");

Distance = int.Parse(ReadLine());//taking distance input

TourAllowances = 5 \* Distance; //5 rupees per km

Telephone\_Allowances = 1000;

GrossSalary=Salary + TourAllowances+Telephone\_Allowances;

}

public void DisplayEmployeeDetails() //implementation of interface method

{

WriteLine("Marketing Executive Details....");

WriteLine("emp\_id: {0}\n emp\_name: {1}\nGross Salary:{2}", emp\_id, emp\_name,GrossSalary);

}

}

internal class Program

{

static void Main(string[] args)

{

Manager mg = new Manager();

MarketingExecutive me = new MarketingExecutive();

WriteLine("Manager Input");

mg.GetSalaryWithEmployeeDetails();

mg.CalculateSalary();

mg.DisplayEmployeeDetails();

WriteLine("Marketing Executive Input");

me.GetSalaryWithEmployeeDetails();

me.CalculateSalary();

me.DisplayEmployeeDetails();

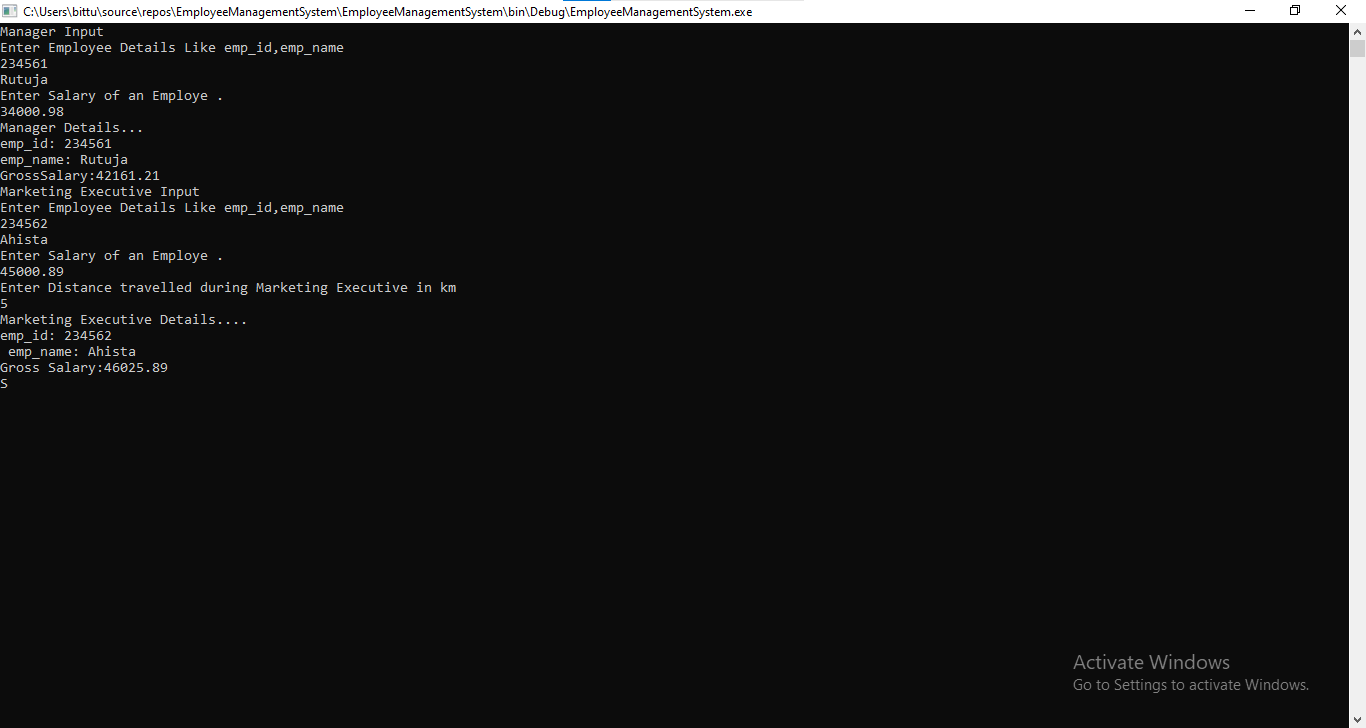
ReadLine();

}

}

}

Output



2.Write a class called MyStack with following members.

* 1. integer array
  2. integer variable to store top position
  3. size of the array.

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

Program

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Console;

using static System.Convert;

namespace MyStackPushPop

{

public class MyStack

{

int size\_of\_stack;

int top\_position = 0;

int[] array = new int[20];

public void GetSizeOfStack()//for getting size of stack

{

WriteLine("Enter Size of Stack");

size\_of\_stack = int.Parse(ReadLine());

}

public void Push()

{

if(top\_position==(size\_of\_stack-1))

{

WriteLine("Stack is Full");

}

else

{

for(int i=0;i<size\_of\_stack;i++)

{

WriteLine("Insert element in stack");

int element = int.Parse(ReadLine());

array[++top\_position] = element;

WriteLine("Item pushed Successfully!");

}

}

}

public object Pop()

{

if(top\_position== -1)

{

WriteLine("Stack is Empty");

return "No elements";

}

else

{

return array[top\_position--];

}

}

public void Display()

{

for(int i = top\_position;i>0;i--)

{

WriteLine("Item {0}: |{1}| ", (i), array[i]);

}

ReadLine();

}

}

internal class Program

{

static void Main(string[] args)

{

MyStack stack = new MyStack();

//stack.GetSizeOfStack();

while(true)

{

Clear();

WriteLine("Stack Menu Program");

WriteLine("1. Enter Size of Stack");

WriteLine("2. Push operation");

WriteLine("3. Pop Operation");

WriteLine("4. Display ");

WriteLine("5. Exit");

Write("Enter Your Choice");

int choice = ToInt32(ReadLine());

switch(choice)

{

case 1:

stack.GetSizeOfStack();

break;

case 2:

stack.Push();

break;

case 3:

WriteLine("Element removed: {0}", stack.Pop());

ReadLine();

break;

case 4:

stack.Display();

break;

case 5:

Environment.Exit(0);

break;

}

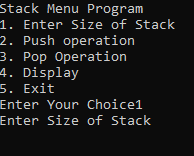
}

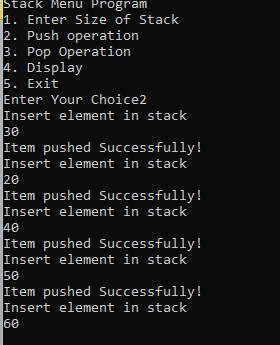
}

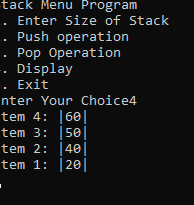
}

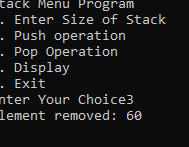
}

Output









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.

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace StackExceptionPro

{

interface Istack

{

Boolean IsEmpty();

void Push(object element);

object Pop();

object Peek();

void Display();

}

class PushPop : Istack

{

public int top;

object[] item;

public int stack\_size;

public PushPop(int stack\_size) //parametrised constructor

{

this.stack\_size = stack\_size;

item = new object[stack\_size];

top = -1;

}

public PushPop() //default constructor

{

stack\_size = 10;

item = new object[stack\_size];

top = -1;

}

public bool IsEmpty() //Method to check is stack empty or full

{

if (top == (stack\_size - 1))

{

return true;

}

else

return false;

}

public void Push(object element)

{

if (top == (stack\_size - 1))

{

throw new Exception("Exception Stack overflow because stack is full");

}

else

{

item[++top] = element;

Console.WriteLine("Item Pushed Successfully");

}

}

public object Pop()

{

if (IsEmpty())

{

throw new Exception("Exception underflow because stack is empty");

return null;

}

else

{

return item[top--];

}

}

public object Peek()

{

if (IsEmpty())

{

throw new Exception("Stack Underflow exception because stack is empty");

return null;

}

else

{

return item[top];

}

}

public void Display()

{

for(int i = top; i>-1;i--)

{

Console.WriteLine(item[i]);

}

}

}

internal class Program

{

static void Main(string[] args)

{

PushPop st = new PushPop();

object element;

while(true)

{

Console.Clear();

Console.WriteLine("1. IsEmpty()");

Console.WriteLine("2. Push()");

Console.WriteLine("3. Pop()");

Console.WriteLine("4. Peek()");

Console.WriteLine("5. Display()");

Console.WriteLine("6. Exit()");

Console.WriteLine("Enter Your Choice");

int choice;

choice = int.Parse(Console.ReadLine());

Console.WriteLine();

switch(choice)

{

case 1:

if(st.IsEmpty())

{

Console.WriteLine("Stack is Empty");

}

else

{

Console.WriteLine("Stack is not empty");

}

break;

case 2:

Console.WriteLine("Enter element");

element = Console.ReadLine();

try

{

st.Push(element);

}

catch(StackOverflowException e)

{

Console.WriteLine(e.Message);

}

break;

case 3:

try

{

element = st.Pop();

Console.WriteLine("Element removed is {0} ", element);

}

catch(Exception e)

{

Console.WriteLine(e.Message);

}

break;

case 4:

try

{

element = st.Peek();

Console.WriteLine("Element removed is {0} ", element);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

break;

case 5:

st.Display();

break;

case 6:

Environment.Exit(0);

break;

default:

Console.WriteLine("Enter valid choice");

break;

}

Console.ReadLine();

}

}

}

OUTPUT

