# Inheritance and Polymorphism

## Objective

* To use inheritance (“is-a” relationship).
* To use polymorphism.
* To create an abstract class
* To use interface.

## 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.

**INPUT:**

using System;

namespace EmployeeManagementSystem

{

public interface IPrintable

{

void DispEmpDetail();

}

public class Employee

{

public int Employeeid;

public string EmployeeName;

public double Salary;

public double GrossSalary;

public void GetSalaryWithEmpDetails()

{

Console.Write("Enter Employee ID:");

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

Console.Write("Enter Employee Name:");

EmployeeName = Console.ReadLine();

Console.Write("Enter Employee Salary:");

Salary = double.Parse(Console.ReadLine());

}

public virtual void CalcSalary()

{

}

}

public class Manager : Employee

{

private double PetrolAllowance;

private double FoodAllowance;

private double OtherAllowance;

private double TDS;

private double NetSalary;

public override void CalcSalary()

{

PetrolAllowance = 8 \* Salary / 100;

FoodAllowance = 13 \* Salary / 100;

OtherAllowance = 3 \* Salary / 100;

GrossSalary = Salary + PetrolAllowance + FoodAllowance + OtherAllowance;

TDS = 18 \* GrossSalary / 100;

NetSalary = GrossSalary - TDS;

Console.WriteLine("Employee Id: {0}\nEmployee Name: {1}\nGross Salary:{2}\nNet Salary:{3}", Employeeid, EmployeeName, GrossSalary,NetSalary);

}

}

public class MarketingExecutive : Employee, IPrintable

{

private int KilometerTravel;

private double TourAllowance;

private double TelephoneAllowances;

public override void CalcSalary()

{

Console.Write("Enter Distance in KM:");

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

TourAllowance = 5 \* KilometerTravel;

TelephoneAllowances = 1000;

GrossSalary = Salary + TourAllowance + TelephoneAllowances;

}

public void DispEmpDetail()

{

Console.WriteLine("Marketing Executive Details");

Console.WriteLine("Employee Id: {0}\nEmployee Name: {1}\nGross Salary:{2}", Employeeid, EmployeeName, GrossSalary);

}

}

class ManagementExc

{

static void Main(string[] args)

{

Manager manager = new Manager();

MarketingExecutive marketingExecutive = new MarketingExecutive();

manager.GetSalaryWithEmpDetails();

manager.CalcSalary();

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("Marketing Executive Input");

marketingExecutive.GetSalaryWithEmpDetails();

marketingExecutive.CalcSalary();

marketingExecutive.DispEmpDetail();

Console.Read();

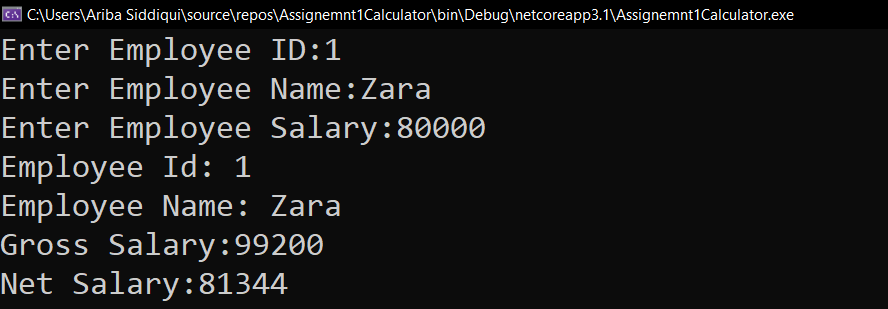
}

}

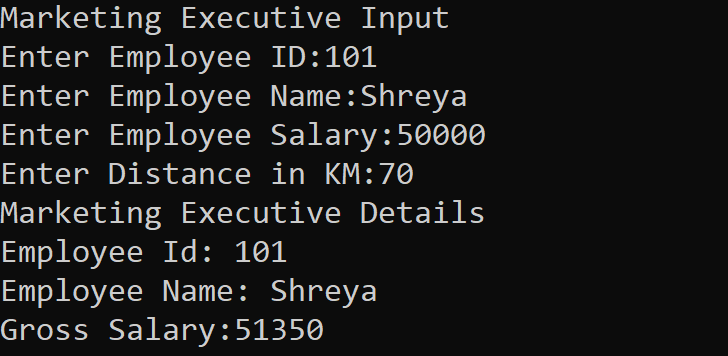
}

**OUTPUT:**

**a)**



**b)**



1. 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.

**Push() and Pop():**

using System;

using System.Collections;

namespace StackAssignment

{

public class MyStack

{

int ArrSize;

int TopPos = 0;

int[] arraystack = new int[10];

public void StackPushPop() {

Stack stack = new Stack();

Console.Write("Enter number of element you want to store in Stack:");

ArrSize = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter {0} Element: ", ArrSize);

//Push Operation

for (TopPos = 0; TopPos < ArrSize; TopPos++)

{

Console.Write("Enter element - {0} : ", TopPos);

stack.Push(Console.ReadLine());

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("Stored Element in Stack");

foreach (var num in stack)

{

Console.WriteLine(num);

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.Write("Before Pop Count in Stack:");

Console.WriteLine(stack.Count);

// Pop operation

while (stack.Count > 0)

{

stack.Pop();

}

Console.Write("After Pop Count in Stack:");

Console.WriteLine(stack.Count);

}

static void Main(string[] args)

{

MyStack myStack = new MyStack();

myStack.StackPushPop();

}

}

}

**Output of Push and Pop Operation:**



**Implementation of ICloneable Interface.**

**INPUT:**

using System;

namespace IcloneableInterfaceExample

{

class CloneAssignment

{

static void Main(string[] args)

{

Console.WriteLine("Iclonable Interface Example");

Console.WriteLine("");

Person person1 = new Person()

{

Name = "Aryan",

Age = 25,

Address = new Address()

{

City = "Navi Mumbai",

Area = "Vashi"

}

};

Person person2 = (Person)person1.Clone();

person2.Name = "Rahul";

person2.Address.City = "Navi Mumbai";

person2.Address.Area = "Belapur";

Console.WriteLine("Person 1 Name: " + person1.Name);

Console.WriteLine("Person 2 Name: "+ person2.Name);

Console.WriteLine("");

Console.WriteLine("Person 1 City: " + person1.Address.City);

Console.WriteLine("Person 1 Area: " + person1.Address.Area);

Console.WriteLine("");

Console.WriteLine("Person 2 City: " + person2.Address.City);

Console.WriteLine("Person 2 Area: " + person2.Address.Area);

}

}

class Person : ICloneable

{

public Person() { }

public Person(Person person)

{

\_name = person.\_name;

\_age = person.\_age;

\_address = (Address)person.\_address.Clone();

}

public Object Clone()

{

return new Person(this);

}

private Address \_address;

public Address Address

{

get { return \_address; }

set { \_address = value; }

}

private string \_name;

public string Name

{

get { return \_name; }

set { \_name = value; }

}

private Int32 \_age;

public Int32 Age

{

get { return \_age; }

set { \_age = value; }

}

}

class Address: ICloneable

{

public Address() { }

public Address(Address address)

{

\_area = address.\_area;

\_city = address.\_city;

}

public Object Clone()

{

return new Address(this);

}

private string \_area;

public string Area

{

get { return \_area; }

set { \_area = value; }

}

private string \_city;

public string City

{

get { return \_city; }

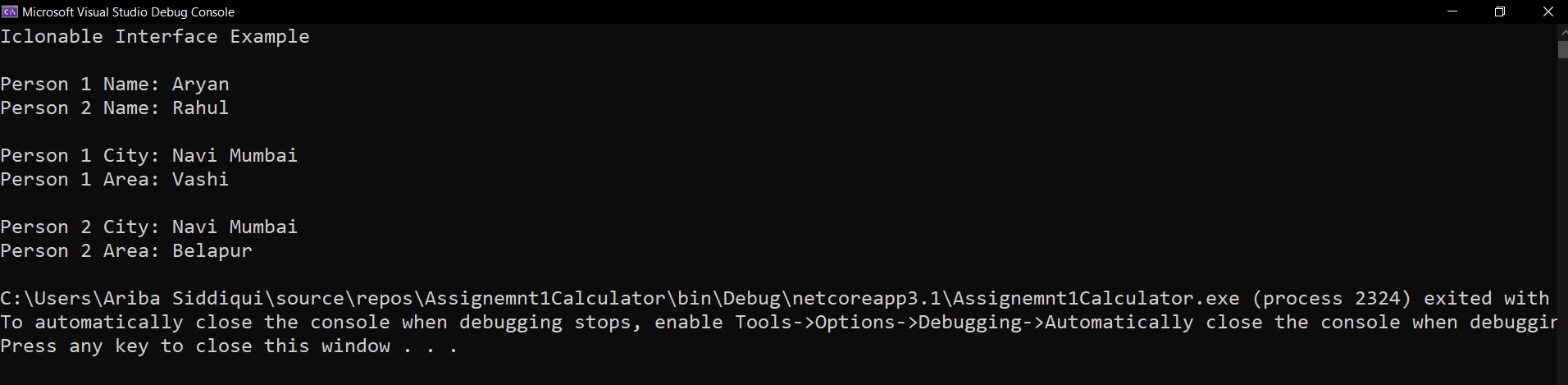
set { \_city = value; }

}

}

}

**OUTPUT:**



1. 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.

**INPUT:**

**Popping from empty array**

using System;

using System.Collections;

class StackException : Exception

{

public StackException() { }

public StackException(string exceptionMessage) : base(exceptionMessage)

{

}

}

public class StackAssignment

{

static void push(int num, ref Stack stack)

{

try

{

if (stack.Count < 5)

{

stack.Push(num);

}

else

{

throw new StackException("StackOverflow Exception: Stack is Full");

}

}

catch (StackException e) // We could be catching anything here

{

Console.WriteLine(e.Message);

}

}

static void pop(ref Stack stack)

{

try

{

if (stack.Count == 0)

{

throw new StackException("popException: Stack is Empty ");

}

else

{

stack.Pop();

}

}

catch (StackException e) // We could be catching anything here

{

Console.WriteLine(e.Message);

}

}

public static void Main(string[] args)

{

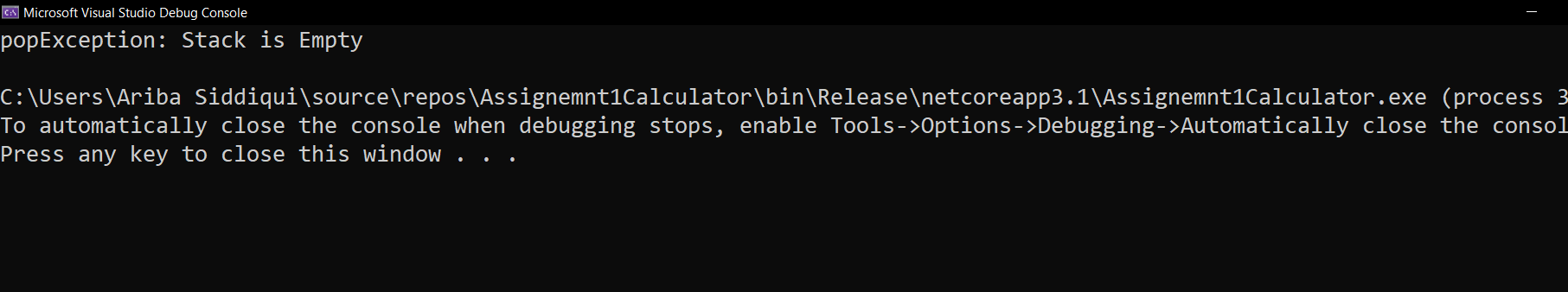
Stack stack = new Stack();

pop(ref stack);

}

}

**OUTPUT:**



**StackOverFlow Exception:**

using System;

using System.Collections;

class StackException : Exception

{

public StackException() { }

public StackException(string exceptionMessage) : base(exceptionMessage)

{

}

}

public class StackAssignment

{

static void push(int num, ref Stack stack)

{

try

{

if (stack.Count < 5)

{

stack.Push(num);

}

else

{

throw new StackException("StackOverflow Exception: Stack is Full");

}

}

catch (StackException e) // We could be catching anything here

{

Console.WriteLine(e.Message);

}

}

static void pop(ref Stack stack)

{

try

{

if (stack.Count == 0)

{

throw new StackException("popException: Stack is Empty ");

}

else

{

stack.Pop();

}

}

catch (StackException e) // We could be catching anything here

{

Console.WriteLine(e.Message);

}

}

public static void Main(string[] args)

{

Stack stack = new Stack();

push(1, ref stack);

push(5, ref stack);

push(3, ref stack);

push(7, ref stack);

push(3, ref stack);

push(77, ref stack);

}

}

**OUTPUT:**

