PRN:-2019033800120821

NAME:-PATEL PRIT SANJAYKUMAR

BATCH: - A

Roll no.: - 412052

**Assignment-6:**

**PART-2:**

**Git repo: https://github.com/pritpatel179/Assignments**

**\*\*Lamda Exp\*\***

**Code:**

using System;

using System.Diagnostics;

namespace LamdaExpExecize

{

public delegate double MaxOftwo(double a,double b);

public class Program {

public static void CallAnonymousMethod()

{

bool positive = new Func<int, bool>(delegate (int int32) { return int32 > 0; })(-11);

new Action<bool>(delegate (bool value) { Console.WriteLine(value); })(positive);

}

public static void CallLambda()

{

bool positive = new Func<int, bool>(int32 => int32 > 0)(1);

new Action<bool>(value =>Console.WriteLine(value))(positive);

}

static void Main()

{

Console.WriteLine("\n1) Anonymous Function and Action Delegate Without Lamda Expression : ");

Program.CallAnonymousMethod();

Console.WriteLine("\n2) Anonymous Function and Action Delegate With Lamda Expression : ");

Program.CallLambda();

//delegate with lamda expression

Console.WriteLine("\n3) Delegate with Lamda Expression : ");

MaxOftwo parse = (double x, double y) => (x > y ? x : y);

Console.WriteLine("\nMax(2.3,1.1) = {0}",parse(2.3,1.1));

Console.WriteLine("\n4) Function Delegate with Lamda Expression : ");

Func<double, double, double> f =(x, y) => { if (x > y) return x; return y; };

double z1 = f(10, 20);

Console.WriteLine("\nMax(10,20) = {0}",z1);

Console.WriteLine("\n5) Function Delegate with DiffrentLamda Expressions : ");

double z2;

Func<double, double, double> f2;

f2 = (x, y) => {

if (x > y)

return x;

return y;

};

z2 = f2(10, 20);

Console.WriteLine("\nMax(10,20) = {0}",z2);

// z holds 20.

f2 = (x, y) => {

if (x < y)

return x;

return y;

};

z2 = f2(10, 20);

Console.WriteLine("Min(10,20) = {0}\n",z2);

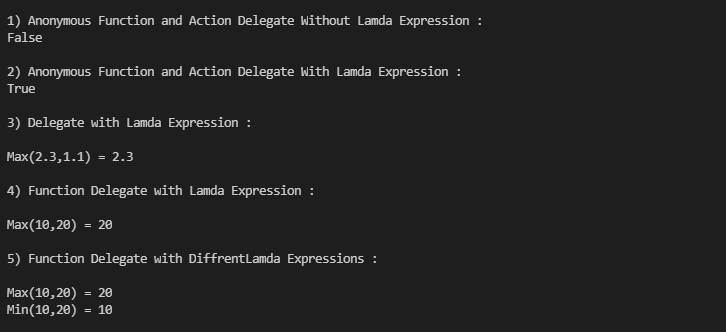
//z2 holds 10.

}

}

}

**Output:**



**\*\*Lamda problem 1\*\***

**Code:**

using System;

using System.Collections.Generic;

using System.Text;

namespace Question\_1

{

class SelectionSort

{

static public void Sort<T>(IList<T>sortArray, Func<T, T, bool>comparision)

{

for (int i = 0; i<sortArray.Count - 1; i++)

{

int key = i;

for (int j = i + 1; j <sortArray.Count; j++)

{

// find minimum element

if (comparision(sortArray[j], sortArray[key]))

{

key = j;

}

}

// swap the smallest element

T temp = sortArray[i];

sortArray[i] = sortArray[key];

sortArray[key] = temp;

}

}

}

public enumTypeOfCake{

Threelayercake=5,

Twolayercake=4,

Normalcake=3,

Pastry =2,

PieceCake=1

}

class Cake

{

int cakeId;

public float price;

public string name;

public TypeOfCaketypeofCake;

public Cake(int CakeId, float Price, string Name, TypeOfCaketpofcake)

{

this.cakeId = CakeId;

this.price = Price;

this.name = Name;

this.typeofCake = tpofcake;

}

internal static bool ComparePrice(Cake c1, Cake c2)

{

return c1.price < c2.price;

}

internal static bool CompareName(Cake c1, Cake c2)

{

if(c1.name.CompareTo(c2.name)== 1)

{

// Console.WriteLine(c1.name.CompareTo(c2.name));

return true;

}

else

{

return false;

}

}

internal static bool CompareTypeOfCake(Cake c1, Cake c2)

{

return c1.typeofCake < c2.typeofCake;

}

}

public class TestGenericMethods

{

public static void Main(string[] args)

{

List<Cake> cakes = new List<Cake>(15);

cakes.Add(new Cake(5, 2900, "BROWNCHOCOLATE", TypeOfCake.Threelayercake));

cakes.Add(new Cake(3, 1459, "REDVALVET", TypeOfCake.Twolayercake));

cakes.Add(new Cake(4, 2489, "PINEAPPLE", TypeOfCake.Normalcake));

cakes.Add(new Cake(1, 149, "ANGLEBYTE", TypeOfCake.PieceCake));

//sorting according to Name of Cakes

Console.WriteLine("\nCakes according to Names : ");

SelectionSort.Sort<Cake>(cakes, Cake.CompareName);

cakes.Reverse();

foreach(Cake c1 in cakes)

{

Console.WriteLine(c1.name);

}

//sorting according to TypeOFCake

Console.WriteLine("\nCakes according to TypeOfCake : ");

SelectionSort.Sort<Cake>(cakes, Cake.CompareTypeOfCake);

foreach(Cake c1 in cakes)

{

Console.WriteLine($"{c1.name} | {c1.typeofCake}");

}

Console.WriteLine("\nSorted array according to price of cakes : ");

SelectionSort.Sort<Cake>(cakes, Cake.ComparePrice);

foreach(Cake c1 in cakes)

{

Console.WriteLine($"{c1.name} | Rs {c1.price}/-");

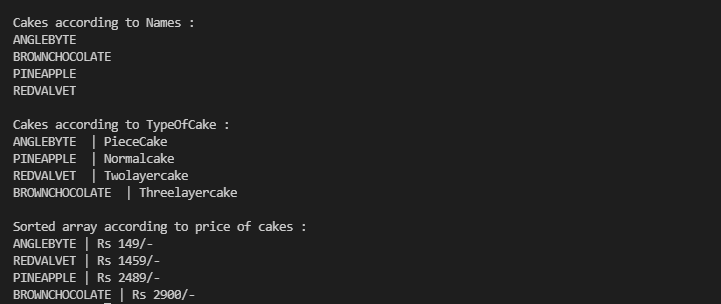
}

}

}

}

**Output:**



**\*\*Lamda problem 2\*\***

**Code:**

using System;

namespace Question\_2

{

class BankCustomer

{

public int accno;

public string acname;

public double bankbalance;

public BankCustomer(int acno,stringacname,double bb)

{

this.accno = acno;

this.acname=acname;

this.bankbalance=bb;

}

}

public delegate void BalanceChecker(double n);

class Program

{

//method for delegate

public static void checkBalance(double balance)

{

if( balance < 0 )

Console.Write("You are overdrawn");

else if(balance < 10)

Console.Write("Your account balance is very low !");

else if(balance < 100)

Console.Write("Watch your spending carefully.");

else

Console.Write("You have over $100 in your account.");

}

static void Main()

{

//instanciate delegate

BalanceChecker checker = checkBalance;

BankCustomer b1= new BankCustomer(123,"ABCD",-5);

Console.Write($"\nHello ,{b1.acname} : ");

checker(b1.bankbalance);

Console.WriteLine($" (Account balance = ${b1.bankbalance})");

BankCustomer b2= new BankCustomer(124,"XYZW",2.5);

Console.Write($"\nHello ,{b2.acname} : ");

checker(b2.bankbalance);

Console.WriteLine($" (Account balance = ${b2.bankbalance})");

BankCustomer b3= new BankCustomer(125,"PQRS",99.9);

Console.Write($"\nHello ,{b3.acname} : ");

checker(b3.bankbalance);

Console.WriteLine($" (Account balance = ${b3.bankbalance})");

BankCustomer b4= new BankCustomer(126,"CVRM",501);

Console.Write($"\nHello ,{b4.acname} : ");

checker(b4.bankbalance);

Console.WriteLine($" (Account balance = ${b4.bankbalance})");

}

}

}

**Output:**

