Assignment No.1

1.Write a console application that swaps two integer values using ref.

using System;

class Program

{

static void Main()

{

int num1 = 10;

int num2 = 20;

Console.WriteLine("Before swapping:");

Console.WriteLine("num1 = " + num1 + ", num2 = " + num2);

Swap(ref num1, ref num2);

Console.WriteLine("After swapping:");

Console.WriteLine("num1 = " + num1 + ", num2 = " + num2);

}

static void Swap(ref int a, ref int b)

{

int temp = a;

a = b;

b = temp;

}

}

2.Write a console application that takes a floating-point number and separates its integer and fractional parts using out.

using System;

class Program

{

static void Main()

{

Console.Write("Enter a floating-point number: ");

float number = float.Parse(Console.ReadLine());

Separate(number, out int integerPart, out float fractionalPart);

Console.WriteLine("Integer part: " + integerPart);

Console.WriteLine("Fractional part: " + fractionalPart);

}

static void Separate(float num, out int integerPart, out float fractionalPart)

{

integerPart = (int)num;

fractionalPart = num - integerPart;

}

}

3.Write a console application that takes two integers and returns their maximum, minimum, and average using out.

using System;

class Program

{

static void Main()

{

Console.Write("Enter first number: ");

int num1 = int.Parse(Console.ReadLine());

Console.Write("Enter second number: ");

int num2 = int.Parse(Console.ReadLine());

GetMaxMinAvg(num1, num2, out int max, out int min, out float avg);

Console.WriteLine("Maximum = " + max);

Console.WriteLine("Minimum = " + min);

Console.WriteLine("Average = " + avg);

}

static void GetMaxMinAvg(int a, int b, out int max, out int min, out float avg)

{

max = (a > b) ? a : b;

min = (a < b) ? a : b;

avg = (a + b) / 2.0f;

}

}

4.Write a console application that finds the minimum number among a variable number of integers using params.

using System;

class Program

{

static void Main()

{

int min = FindMin(5, 3, 8, 1, 6);

Console.WriteLine("Minimum number is: " + min);

}

static int FindMin(params int[] numbers)

{

int min = numbers[0];

foreach (int num in numbers)

{

if (num < min)

min = num;

}

return min;

}

}

5.Write a console application that takes multiple student names and displays greetings for each using params. The method signature should be: void greetings(string msg, params string[] students)

using System;

class Program

{

static void Main()

{

greetings("Welcome", "Alice", "Bob", "Charlie");

}

static void greetings(string msg, params string[] students)

{

foreach (string student in students)

{

Console.WriteLine(msg + ", " + student + "!");

}

}

}

Assignment No. 2

1.Define class Student (RollNo,Name,Address,MobileNo). Define two method accept and display to read and show student information. Create ‘n’ objects of Student class. Call accept and display methods for each object.

using System;

class Student

{

public int RollNo;

public string Name;

public string Address;

public string MobileNo;

// Method to accept student details

public void Accept()

{

Console.Write("Enter Roll No: ");

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

Console.Write("Enter Name: ");

Name = Console.ReadLine();

Console.Write("Enter Address: ");

Address = Console.ReadLine();

Console.Write("Enter Mobile No: ");

MobileNo = Console.ReadLine();

}

// Method to display student details

public void Display()

{

Console.WriteLine("\n--- Student Details ---");

Console.WriteLine("Roll No: " + RollNo);

Console.WriteLine("Name: " + Name);

Console.WriteLine("Address: " + Address);

Console.WriteLine("Mobile No: " + MobileNo);

}

}

class Program

{

static void Main()

{

Console.Write("Enter number of students: ");

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

Student[] students = new Student[n];

// Accept details for each student

for (int i = 0; i < n; i++)

{

Console.WriteLine($"\nEnter details for Student {i + 1}:");

students[i] = new Student();

students[i].Accept();

}

// Display details for each student

for (int i = 0; i < n; i++)

{

Console.WriteLine($"\nDetails of Student {i + 1}:");

students[i].Display();

}

}

}

2.Define class Employee(Id,Name,Address,Salary). Write default and Parameterised constructors. also write display method to display Employee information.

using System;

class Employee

{

private int Id;

private string Name;

private string Address;

private double Salary;

// Default Constructor

public Employee()

{

Id = 0;

Name = "Unknown";

Address = "Not Provided";

Salary = 0.0;

}

// Parameterized Constructor

public Employee(int id, string name, string address, double salary)

{

Id = id;

Name = name;

Address = address;

Salary = salary;

}

// Method to display employee details

public void Display()

{

Console.WriteLine("\n--- Employee Details ---");

Console.WriteLine("Id: " + Id);

Console.WriteLine("Name: " + Name);

Console.WriteLine("Address: " + Address);

Console.WriteLine("Salary: " + Salary);

}

}

class Program

{

static void Main()

{

// Create object using default constructor

Employee emp1 = new Employee();

Console.WriteLine("Details of Employee 1 (default constructor):");

emp1.Display();

// Create object using parameterized constructor

Employee emp2 = new Employee(201, "John Doe", "Pune", 50000);

Console.WriteLine("\nDetails of Employee 2 (parameterized constructor):");

emp2.Display();

}

}

3. Define a class Account with private members -AccId,AccName,AccBalance. Define Property for AccBalance. Also define read only property for AccId. Display Account information

using System;

class Account

{

private int accId;

private string accName;

private double accBalance;

// Constructor to initialize account details

public Account(int id, string name, double balance)

{

accId = id;

accName = name;

accBalance = balance;

}

// Read-only property for AccId

public int AccId

{

get { return accId; }

}

// Property for AccBalance

public double AccBalance

{

get { return accBalance; }

set { accBalance = value; }

}

// Property for AccName

public string AccName

{

get { return accName; }

set { accName = value; }

}

// Method to display account details

public void Display()

{

Console.WriteLine("\n--- Account Details ---");

Console.WriteLine("Account ID: " + AccId);

Console.WriteLine("Account Name: " + AccName);

Console.WriteLine("Account Balance: " + AccBalance);

}

}

class Program

{

static void Main()

{

// Create account object

Account acc = new Account(101, "Alice", 10000);

// Display initial details

Console.WriteLine("Initial Account Details:");

acc.Display();

// Update balance and name

acc.AccBalance += 5000;

acc.AccName = "Alice Johnson";

// Display updated details

Console.WriteLine("\nUpdated Account Details:");

acc.Display();

}

}

4. Define class Item that will contain string array of as ItemName. Define Indexer to access ItemName of Particular Id.

using System;

class Item

{

private string[] itemName;

// Constructor to initialize the array

public Item(int size)

{

itemName = new string[size];

}

// Indexer to access elements like an array

public string this[int index]

{

get

{

if (index >= 0 && index < itemName.Length)

return itemName[index];

else

return "Invalid Index";

}

set

{

if (index >= 0 && index < itemName.Length)

itemName[index] = value;

}

}

// Method to display all items

public void DisplayItems()

{

Console.WriteLine("\n--- Items List ---");

for (int i = 0; i < itemName.Length; i++)

{

Console.WriteLine($"Item[{i}]: {itemName[i]}");

}

}

}

class Program

{

static void Main()

{

Console.Write("Enter the number of items: ");

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

Item items = new Item(size);

// Accept items using indexer

for (int i = 0; i < size; i++)

{

Console.Write($"Enter name for item {i}: ");

items[i] = Console.ReadLine();

}

// Display items using indexer

items.DisplayItems();

}

}

Assignment No.3

1. Write a console application that contains unicast delegate to call the following method :

(i) int Prime(int n) (ii) int reverse(int n) (iii) int sum\_digit(int n) (iv) int Fact(int n)

using System;

// Define the delegate type

public delegate int MyDelegate(int n);

class Program

{

static void Main()

{

Console.Write("Enter a number: ");

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

// Delegate pointing to Prime method

MyDelegate primeDelegate = new MyDelegate(Prime);

int isPrime = primeDelegate(num);

Console.WriteLine(isPrime == 1 ? "Prime number" : "Not a prime number");

// Delegate pointing to Reverse method

MyDelegate reverseDelegate = new MyDelegate(Reverse);

int reversed = reverseDelegate(num);

Console.WriteLine("Reversed number: " + reversed);

// Delegate pointing to Sum\_Digit method

MyDelegate sumDelegate = new MyDelegate(Sum\_Digit);

int sum = sumDelegate(num);

Console.WriteLine("Sum of digits: " + sum);

// Delegate pointing to Fact method

MyDelegate factDelegate = new MyDelegate(Fact);

int factorial = factDelegate(num);

Console.WriteLine("Factorial: " + factorial);

}

// Method to check if number is prime

static int Prime(int n)

{

if (n <= 1) return 0;

for (int i = 2; i <= Math.Sqrt(n); i++)

{

if (n % i == 0)

return 0;

}

return 1;

}

// Method to reverse the digits of number

static int Reverse(int n)

{

int rev = 0;

while (n > 0)

{

rev = rev \* 10 + n % 10;

n /= 10;

}

return rev;

}

// Method to sum the digits of number

static int Sum\_Digit(int n)

{

int sum = 0;

while (n > 0)

{

sum += n % 10;

n /= 10;

}

return sum;

}

// Method to find factorial

static int Fact(int n)

{

if (n == 0 || n == 1)

return 1;

int fact = 1;

for (int i = 2; i <= n; i++)

{

fact \*= i;

}

return fact;

}

}

2. Write a console application that contains multicast delegate for the following method :

(i) void Slength(String s); (ii) void StoUpper(String s); (iii) void SChangedCase(String s);

(iv) void Sreverse(String s); (v) void CountVowels(String s);

using System;

// Define a delegate that takes a string and returns void

public delegate void StringDelegate(string str);

class Program

{

static void Main()

{

Console.Write("Enter a string: ");

string input = Console.ReadLine();

// Create a multicast delegate

StringDelegate operations = Length;

operations += ToUpperCase;

operations += ToLowerCase;

operations += ReverseString;

// Call all methods through the multicast delegate

operations(input);

}

// Method to print length of the string

static void Length(string str)

{

Console.WriteLine("Length of string: " + str.Length);

}

// Method to convert string to uppercase

static void ToUpperCase(string str)

{

Console.WriteLine("Uppercase: " + str.ToUpper());

}

// Method to convert string to lowercase

static void ToLowerCase(string str)

{

Console.WriteLine("Lowercase: " + str.ToLower());

}

// Method to reverse the string

static void ReverseString(string str)

{

char[] chars = str.ToCharArray();

Array.Reverse(chars);

Console.WriteLine("Reversed string: " + new string(chars));

}

}

3. Create a class called WeatherStation. It has an event named OnTemperatureAlert that uses a delegate called TemperatureAlertHandler. When the temperature goes above a set limit, two handlers DisplayDevice and CoolingSystem are called.

using System;

// Define delegate for the event

public delegate void TemperatureExceededHandler(int temperature);

// Publisher class

class WeatherStation

{

public event TemperatureExceededHandler TemperatureExceeded;

private int threshold;

public WeatherStation(int limit)

{

threshold = limit;

}

public void CheckTemperature(int currentTemp)

{

Console.WriteLine("Checking temperature: " + currentTemp + "°C");

if (currentTemp > threshold)

{

Console.WriteLine("Temperature exceeded limit!");

OnTemperatureExceeded(currentTemp); // Raise the event

}

else

{

Console.WriteLine("Temperature is within safe limit.");

}

}

// Method to raise the event

protected virtual void OnTemperatureExceeded(int temp)

{

if (TemperatureExceeded != null)

{

TemperatureExceeded(temp);

}

}

}

// Subscriber class or method

class TemperatureAlert

{

public void Alert(int temp)

{

Console.WriteLine("ALERT! Temperature is too high: " + temp + "°C");

}

}

// Main program

class Program

{

static void Main()

{

WeatherStation station = new WeatherStation(40);

TemperatureAlert alert = new TemperatureAlert();

// Subscribe to the event

station.TemperatureExceeded += alert.Alert;

// Simulate temperature readings

int[] readings = { 35, 42, 38, 45 };

foreach (int temp in readings)

{

station.CheckTemperature(temp);

Console.WriteLine();

}

}

}

Assignment No. 4

1. Define a class Employee having private members – id, name, department, salary. Define default and parameterized constructors. Create a subclass called “Manager” with private member bonus. Define methods accept and display in both the classes. Create n objects of the Manager class and display the details of the manager having the maximum total salary (salary+bonus).

using System;

// Base class

class Employee

{

private int id;

private string name;

private string department;

private double salary;

// Default constructor

public Employee()

{

id = 0;

name = "Unknown";

department = "None";

salary = 0.0;

}

// Parameterized constructor

public Employee(int id, string name, string department, double salary)

{

this.id = id;

this.name = name;

this.department = department;

this.salary = salary;

}

// Method to accept data

public virtual void Accept()

{

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

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

Console.Write("Enter Name: ");

name = Console.ReadLine();

Console.Write("Enter Department: ");

department = Console.ReadLine();

Console.Write("Enter Salary: ");

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

}

// Method to display data

public virtual void Display()

{

Console.WriteLine("ID: " + id);

Console.WriteLine("Name: " + name);

Console.WriteLine("Department: " + department);

Console.WriteLine("Salary: " + salary);

}

// Provide access to salary for derived classes

public double GetSalary()

{

return salary;

}

// Provide access to other details if needed

public int GetId()

{

return id;

}

public string GetName()

{

return name;

}

public string GetDepartment()

{

return department;

}

}

// Derived class

class Manager : Employee

{

private double bonus;

// Default constructor

public Manager() : base()

{

bonus = 0.0;

}

// Parameterized constructor

public Manager(int id, string name, string department, double salary, double bonus)

: base(id, name, department, salary)

{

this.bonus = bonus;

}

// Override Accept method

public override void Accept()

{

base.Accept();

Console.Write("Enter Bonus: ");

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

}

// Override Display method

public override void Display()

{

base.Display();

Console.WriteLine("Bonus: " + bonus);

Console.WriteLine("Total Salary (Salary + Bonus): " + (GetSalary() + bonus));

}

// Method to get total salary

public double GetTotalSalary()

{

return GetSalary() + bonus;

}

}

class Program

{

static void Main()

{

Console.Write("Enter number of managers: ");

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

Manager[] managers = new Manager[n];

// Accept details for each manager

for (int i = 0; i < n; i++)

{

Console.WriteLine($"\nEnter details for Manager {i + 1}:");

managers[i] = new Manager();

managers[i].Accept();

}

// Find manager with maximum total salary

Manager maxManager = managers[0];

for (int i = 1; i < n; i++)

{

if (managers[i].GetTotalSalary() > maxManager.GetTotalSalary())

{

maxManager = managers[i];

}

}

// Display manager with max total salary

Console.WriteLine("\nManager with Maximum Total Salary:");

maxManager.Display();

}

}

2. Define an interface “StackOperations” which declares methods for a static stack. Define a class

“MyStack” which contains an array and top as data members and implements the above interface.

Initialize the stack using a constructor. Write a menu driven program to perform operations on a

stack object.

using System;

// Define the interface for stack operations

interface StackOperations

{

void Push(int item);

int Pop();

int Peek();

void Display();

}

// Implement the stack in MyStack class

class MyStack : StackOperations

{

private int[] stackArray;

private int top;

private int maxSize;

// Constructor to initialize the stack

public MyStack(int size)

{

maxSize = size;

stackArray = new int[maxSize];

top = -1;

}

// Push operation

public void Push(int item)

{

if (top >= maxSize - 1)

{

Console.WriteLine("Stack Overflow! Cannot push item.");

return;

}

stackArray[++top] = item;

Console.WriteLine($"{item} pushed into the stack.");

}

// Pop operation

public int Pop()

{

if (top < 0)

{

Console.WriteLine("Stack Underflow! Cannot pop item.");

return -1;

}

int item = stackArray[top--];

Console.WriteLine($"{item} popped from the stack.");

return item;

}

// Peek operation

public int Peek()

{

if (top < 0)

{

Console.WriteLine("Stack is empty!");

return -1;

}

Console.WriteLine($"Top element is {stackArray[top]}");

return stackArray[top];

}

// Display operation

public void Display()

{

if (top < 0)

{

Console.WriteLine("Stack is empty!");

return;

}

Console.WriteLine("Stack elements are:");

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

{

Console.WriteLine(stackArray[i]);

}

}

}

class Program

{

static void Main()

{

Console.Write("Enter the size of the stack: ");

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

MyStack stack = new MyStack(size);

int choice;

do

{

Console.WriteLine("\n--- Stack Operations Menu ---");

Console.WriteLine("1. Push");

Console.WriteLine("2. Pop");

Console.WriteLine("3. Peek");

Console.WriteLine("4. Display");

Console.WriteLine("5. Exit");

Console.Write("Enter your choice: ");

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

switch (choice)

{

case 1:

Console.Write("Enter element to push: ");

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

stack.Push(item);

break;

case 2:

stack.Pop();

break;

case 3:

stack.Peek();

break;

case 4:

stack.Display();

break;

case 5:

Console.WriteLine("Exiting...");

break;

default:

Console.WriteLine("Invalid choice! Try again.");

break;

}

} while (choice != 5);

}

}

Assignment No. 5

1.Write a class Maximum to find the maximum between two , three and four numbers using method overloading

using System;

class Maximum

{

// Find maximum between two numbers

public int FindMax(int a, int b)

{

return (a > b) ? a : b;

}

// Find maximum between three numbers

public int FindMax(int a, int b, int c)

{

int max = (a > b) ? a : b;

return (max > c) ? max : c;

}

// Find maximum between four numbers

public int FindMax(int a, int b, int c, int d)

{

int max = FindMax(a, b, c); // reuse three-number version

return (max > d) ? max : d;

}

}

class Program

{

static void Main()

{

Maximum max = new Maximum();

// Find maximum between two numbers

Console.Write("Enter first number: ");

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

Console.Write("Enter second number: ");

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

Console.WriteLine($"Maximum between {a} and {b} is: {max.FindMax(a, b)}");

// Find maximum between three numbers

Console.Write("\nEnter first number: ");

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

Console.Write("Enter second number: ");

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

Console.Write("Enter third number: ");

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

Console.WriteLine($"Maximum between {c}, {d}, and {e} is: {max.FindMax(c, d, e)}");

// Find maximum between four numbers

Console.Write("\nEnter first number: ");

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

Console.Write("Enter second number: ");

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

Console.Write("Enter third number: ");

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

Console.Write("Enter fourth number: ");

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

Console.WriteLine($"Maximum between {f}, {g}, {h}, and {i} is: {max.FindMax(f, g, h, i)}");

}

}

2.Create an abstract class Shape with methods calc\_area and calc\_volume. Derive three classes

Sphere(radius) , Cone(radius, height) and Cylinder(radius, height), Box(length, breadth,

height) from it. Calculate area and volume of all. (Use Method overriding)

using System;

// Abstract base class

abstract class Shape

{

public abstract double calc\_area();

public abstract double calc\_volume();

}

// Sphere

class Sphere : Shape

{

private double radius;

public Sphere(double radius)

{

this.radius = radius;

}

public override double calc\_area()

{

return 4 \* Math.PI \* radius \* radius;

}

public override double calc\_volume()

{

return (4.0 / 3.0) \* Math.PI \* Math.Pow(radius, 3);

}

}

// Cone

class Cone : Shape

{

private double radius, height;

public Cone(double radius, double height)

{

this.radius = radius;

this.height = height;

}

public override double calc\_area()

{

double slant = Math.Sqrt(height \* height + radius \* radius);

return Math.PI \* radius \* (radius + slant);

}

public override double calc\_volume()

{

return (1.0 / 3.0) \* Math.PI \* radius \* radius \* height;

}

}

// Cylinder

class Cylinder : Shape

{

private double radius, height;

public Cylinder(double radius, double height)

{

this.radius = radius;

this.height = height;

}

public override double calc\_area()

{

return 2 \* Math.PI \* radius \* (radius + height);

}

public override double calc\_volume()

{

return Math.PI \* radius \* radius \* height;

}

}

// Box (Cuboid)

class Box : Shape

{

private double length, breadth, height;

public Box(double length, double breadth, double height)

{

this.length = length;

this.breadth = breadth;

this.height = height;

}

public override double calc\_area()

{

return 2 \* (length \* breadth + breadth \* height + length \* height);

}

public override double calc\_volume()

{

return length \* breadth \* height;

}

}

class Program

{

static void Main()

{

// Sphere

Sphere s = new Sphere(5);

Console.WriteLine("Sphere -> Area: " + s.calc\_area().ToString("F2") +

", Volume: " + s.calc\_volume().ToString("F2"));

// Cone

Cone c = new Cone(3, 7);

Console.WriteLine("Cone -> Area: " + c.calc\_area().ToString("F2") +

", Volume: " + c.calc\_volume().ToString("F2"));

// Cylinder

Cylinder cy = new Cylinder(4, 10);

Console.WriteLine("Cylinder -> Area: " + cy.calc\_area().ToString("F2") +

", Volume: " + cy.calc\_volume().ToString("F2"));

// Box

Box b = new Box(2, 3, 4);

Console.WriteLine("Box -> Area: " + b.calc\_area().ToString("F2") +

", Volume: " + b.calc\_volume().ToString("F2"));

}

}

3. WAP on class Fraction with properties numerator and denominator write appropriate

constructor and display method for the Class. Overload following Operator: (+,-,\*,/)

using System;

class Fraction

{

public int Numerator { get; set; }

public int Denominator { get; set; }

// Default constructor

public Fraction()

{

Numerator = 0;

Denominator = 1;

}

// Parameterized constructor

public Fraction(int numerator, int denominator)

{

if (denominator == 0)

throw new ArgumentException("Denominator cannot be zero.");

Numerator = numerator;

Denominator = denominator;

Simplify();

}

// Method to display fraction

public void Display()

{

Console.WriteLine($"{Numerator}/{Denominator}");

}

// Simplify fraction

private void Simplify()

{

int gcd = GCD(Math.Abs(Numerator), Math.Abs(Denominator));

Numerator /= gcd;

Denominator /= gcd;

// Handle negative denominator

if (Denominator < 0)

{

Numerator = -Numerator;

Denominator = -Denominator;

}

}

// GCD Helper Method

private int GCD(int a, int b)

{

while (b != 0)

{

int temp = b;

b = a % b;

a = temp;

}

return a;

}

// Operator Overloading

public static Fraction operator +(Fraction a, Fraction b)

{

return new Fraction(

a.Numerator \* b.Denominator + b.Numerator \* a.Denominator,

a.Denominator \* b.Denominator

);

}

public static Fraction operator -(Fraction a, Fraction b)

{

return new Fraction(

a.Numerator \* b.Denominator - b.Numerator \* a.Denominator,

a.Denominator \* b.Denominator

);

}

public static Fraction operator \*(Fraction a, Fraction b)

{

return new Fraction(

a.Numerator \* b.Numerator,

a.Denominator \* b.Denominator

);

}

public static Fraction operator /(Fraction a, Fraction b)

{

if (b.Numerator == 0)

throw new DivideByZeroException("Cannot divide by zero fraction.");

return new Fraction(

a.Numerator \* b.Denominator,

a.Denominator \* b.Numerator

);

}

}

class Program

{

static void Main()

{

Fraction f1 = new Fraction(1, 2); // 1/2

Fraction f2 = new Fraction(3, 4); // 3/4

Console.Write("f1 = "); f1.Display();

Console.Write("f2 = "); f2.Display();

Fraction sum = f1 + f2;

Console.Write("f1 + f2 = "); sum.Display();

Fraction diff = f1 - f2;

Console.Write("f1 - f2 = "); diff.Display();

Fraction product = f1 \* f2;

Console.Write("f1 \* f2 = "); product.Display();

Fraction quotient = f1 / f2;

Console.Write("f1 / f2 = "); quotient.Display();

}

}

Assignment No. 6

1.Create a class Student (SName,Address,BirthDate) . Write accept and display methods.Throw a user defined Exception “InvalidBirthdate” when the age of Student is not between

18 to 25.

using System;

// User-defined Exception

class InvalidBirthdateException : Exception

{

public InvalidBirthdateException(string message) : base(message)

{

}

}

// Student class

class Student

{

private string SName;

private string Address;

private DateTime BirthDate;

// Accept student details

public void Accept()

{

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

SName = Console.ReadLine();

Console.Write("Enter Address: ");

Address = Console.ReadLine();

Console.Write("Enter Birthdate (yyyy-mm-dd): ");

BirthDate = DateTime.Parse(Console.ReadLine());

int age = CalculateAge(BirthDate);

// Validate age

if (age < 18 || age > 25)

{

throw new InvalidBirthdateException(

$"Invalid Age: {age}. Age must be between 18 and 25."

);

}

}

// Display student details

public void Display()

{

Console.WriteLine("\n--- Student Details ---");

Console.WriteLine("Name: " + SName);

Console.WriteLine("Address: " + Address);

Console.WriteLine("Birthdate: " + BirthDate.ToString("yyyy-MM-dd"));

Console.WriteLine("Age: " + CalculateAge(BirthDate));

}

// Helper method to calculate age

private int CalculateAge(DateTime birthDate)

{

DateTime today = DateTime.Today;

int age = today.Year - birthDate.Year;

if (birthDate.Date > today.AddYears(-age)) age--;

return age;

}

}

class Program

{

static void Main()

{

try

{

Student s = new Student();

s.Accept();

s.Display();

}

catch (InvalidBirthdateException ex)

{

Console.WriteLine("\nException: " + ex.Message);

}

catch (Exception ex)

{

Console.WriteLine("\nGeneral Error: " + ex.Message);

}

}

}

2.Write a program in C# to throw & handle following exceptions in bankingl applications :

i) MinimumBalanceException : When balance is less than 1000

ii) DailyDepositException : In a day ,only one lakh can be deposited

iii) Display details of each exception. Use required members and methods.

using System;

// Custom Exception for Minimum Balance

class MinimumBalanceException : Exception

{

public MinimumBalanceException(string message) : base(message) { }

}

// Custom Exception for Daily Deposit Limit

class DailyDepositException : Exception

{

public DailyDepositException(string message) : base(message) { }

}

// BankAccount Class

class BankAccount

{

public int AccountNumber { get; set; }

public string HolderName { get; set; }

public double Balance { get; private set; }

public BankAccount(int accNo, string name, double initialBalance)

{

AccountNumber = accNo;

HolderName = name;

Balance = initialBalance;

}

// Deposit Method

public void Deposit(double amount)

{

if (amount > 100000)

{

throw new DailyDepositException(

$"Deposit failed! You tried to deposit ₹{amount}. Daily deposit limit is ₹100000."

);

}

Balance += amount;

Console.WriteLine($"₹{amount} deposited successfully! Current Balance: ₹{Balance}");

}

// Withdraw Method

public void Withdraw(double amount)

{

if (Balance - amount < 1000)

{

throw new MinimumBalanceException(

$"Withdrawal failed! After withdrawing ₹{amount}, balance would fall below ₹1000."

);

}

Balance -= amount;

Console.WriteLine($"₹{amount} withdrawn successfully! Current Balance: ₹{Balance}");

}

// Display Account Info

public void Display()

{

Console.WriteLine("\n--- Account Details ---");

Console.WriteLine($"Account Number: {AccountNumber}");

Console.WriteLine($"Holder Name: {HolderName}");

Console.WriteLine($"Balance: ₹{Balance}");

}

}

// Main Program

class Program

{

static void Main()

{

BankAccount account = new BankAccount(101, "Alice", 5000);

try

{

account.Display();

// Valid Deposit

account.Deposit(50000);

// Invalid Deposit

account.Deposit(150000);

}

catch (DailyDepositException ex)

{

Console.WriteLine("\nException: " + ex.Message);

}

try

{

// Valid Withdraw

account.Withdraw(3000);

// Invalid Withdraw

account.Withdraw(2000);

}

catch (MinimumBalanceException ex)

{

Console.WriteLine("\nException: " + ex.Message);

}

}

}

Assignment No. 7

1.Accept ‘n’ integers from the user and store them in a collection. Display them in the sorted order. The collection should not accept duplicate elements. (Use a suitable collection).

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

Console.Write("Enter number of elements: ");

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

SortedSet<int> numbers = new SortedSet<int>();

Console.WriteLine("Enter the numbers:");

for (int i = 0; i < n; i++)

{

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

if (!numbers.Add(num)) // Add() returns false if duplicate

{

Console.WriteLine($"Duplicate {num} ignored!");

}

}

Console.WriteLine("\nNumbers in sorted order (no duplicates):");

foreach (int num in numbers)

{

Console.Write(num + " ");

}

}

}

2. Construct linked List containing names of colors: red, blue, yellow and orange. Then extend your

program to do the following:

i. Display the contents of the List using an Iterator;

ii. Display the contents of the List in reverse order using a ListIterator

iii. Create another list containing pink and green. Insert the elements of this list between blue and

yellow.

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

// Step 1: Create linked list with colors

LinkedList<string> colors = new LinkedList<string>();

colors.AddLast("red");

colors.AddLast("blue");

colors.AddLast("yellow");

colors.AddLast("orange");

Console.WriteLine("Original List:");

foreach (string color in colors)

{

Console.Write(color + " ");

}

// (i) Display contents using iterator

Console.WriteLine("\n\nList using Iterator:");

IEnumerator<string> iterator = colors.GetEnumerator();

while (iterator.MoveNext())

{

Console.Write(iterator.Current + " ");

}

// (ii) Display contents in reverse order

Console.WriteLine("\n\nList in Reverse Order:");

LinkedListNode<string> node = colors.Last;

while (node != null)

{

Console.Write(node.Value + " ");

node = node.Previous;

}

// (iii) Insert another list (pink, green) between blue and yellow

LinkedList<string> newColors = new LinkedList<string>();

newColors.AddLast("pink");

newColors.AddLast("green");

LinkedListNode<string> blueNode = colors.Find("blue");

if (blueNode != null)

{

// Insert after blue

foreach (string newColor in newColors)

{

colors.AddAfter(blueNode, newColor);

blueNode = blueNode.Next; // move forward

}

}

Console.WriteLine("\n\nList after inserting new colors:");

foreach (string color in colors)

{

Console.Write(color + " ");

}

}

}

3. Create a Hash table containing student name and percentage. Display the details of the hash table.

Also search for a specific student and display percentage of that student.

using System;

using System.Collections;

class Program

{

static void Main()

{

// Step 1: Create a Hashtable

Hashtable students = new Hashtable();

// Adding student name and percentage

students.Add("Alice", 85.5);

students.Add("Bob", 92.3);

students.Add("Charlie", 76.8);

students.Add("David", 88.9);

// Step 2: Display all details

Console.WriteLine("Student Records:");

foreach (DictionaryEntry entry in students)

{

Console.WriteLine("Name: " + entry.Key + " | Percentage: " + entry.Value);

}

// Step 3: Search for a student

Console.Write("\nEnter student name to search: ");

string name = Console.ReadLine();

if (students.ContainsKey(name))

{

Console.WriteLine($"Percentage of {name}: {students[name]}");

}

else

{

Console.WriteLine("Student not found!");

}

}

}

4. Assuming one already has a stack library with all six operations available, implement the stack

operation that deletes the bottom (not the top) element of the stack using available stack operations?

using System;

using System.Collections.Generic;

class Program

{

// Function to delete the bottom element

static void DeleteBottom(Stack<int> stack)

{

if (stack.Count == 0)

return;

int top = stack.Pop();

if (stack.Count == 0)

{

// This is the bottom element → discard

return;

}

// Recursive call

DeleteBottom(stack);

// Push back other elements

stack.Push(top);

}

static void Main()

{

Stack<int> stack = new Stack<int>();

// Push elements

stack.Push(10);

stack.Push(20);

stack.Push(30);

stack.Push(40);

stack.Push(50);

Console.WriteLine("Original Stack: " + string.Join(", ", stack));

DeleteBottom(stack);

Console.WriteLine("After deleting bottom element: " + string.Join(", ", stack));

}

}

5. Implement Hash table to store ‘n’ records of students (Name, Percentage). Write a menu driven

program to :

1. Add student 2. Display details of all students

3. Search student 4. Find out highest marks

using System;

using System.Collections;

class Program

{

static void Main()

{

Hashtable students = new Hashtable();

while (true)

{

Console.WriteLine("\n--- Student Menu ---");

Console.WriteLine("1. Add Student");

Console.WriteLine("2. Display All Students");

Console.WriteLine("3. Search Student");

Console.WriteLine("4. Find Student with Highest Marks");

Console.WriteLine("5. Exit");

Console.Write("Enter your choice: ");

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

switch (choice)

{

case 1:

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

string name = Console.ReadLine();

Console.Write("Enter Percentage: ");

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

if (!students.ContainsKey(name))

{

students.Add(name, percentage);

Console.WriteLine("Student added successfully!");

}

else

{

Console.WriteLine("Student already exists!");

}

break;

case 2:

Console.WriteLine("\n--- All Students ---");

if (students.Count == 0)

Console.WriteLine("No records found.");

else

{

foreach (DictionaryEntry entry in students)

{

Console.WriteLine($"Name: {entry.Key}, Percentage: {entry.Value}");

}

}

break;

case 3:

Console.Write("Enter Student Name to Search: ");

string searchName = Console.ReadLine();

if (students.ContainsKey(searchName))

Console.WriteLine($"Percentage of {searchName}: {students[searchName]}");

else

Console.WriteLine("Student not found!");

break;

case 4:

Console.WriteLine("\n--- Student with Highest Marks ---");

if (students.Count == 0)

Console.WriteLine("No records available.");

else

{

string topStudent = "";

double maxMarks = double.MinValue;

foreach (DictionaryEntry entry in students)

{

if ((double)entry.Value > maxMarks)

{

maxMarks = (double)entry.Value;

topStudent = (string)entry.Key;

}

}

Console.WriteLine($"Topper: {topStudent}, Percentage: {maxMarks}");

}

break;

case 5:

return; // exit program

default:

Console.WriteLine("Invalid choice! Please try again.");

break;

}

}

}

}

6.Create two singly linked lists for string data (Do not allow duplicate elements in

one list). Perform

a. Union b. Intersection c. Combining corresponding elements of the lists into a new list (only if they are of the same size)

using System;

using System.Collections.Generic;

using System.Linq;

public class LinkedListOperations

{

public static void Main(string[] args)

{

// 1. Create a collection for string data that does not allow duplicates.

// HashSet<T> is ideal as it ensures all elements are unique.

HashSet<string> uniqueList = new HashSet<string>();

uniqueList.Add("Apple");

uniqueList.Add("Banana");

uniqueList.Add("Cherry");

// Attempting to add a duplicate element. It will be ignored.

bool added = uniqueList.Add("Apple");

// 'added' will be false, and the set remains unchanged.

Console.WriteLine("List 1 (Unique Elements - HashSet):");

Console.WriteLine(string.Join(" -> ", uniqueList));

Console.WriteLine("-----------------------------------");

// 2. Create a standard singly linked list for string data.

LinkedList<string> linkedList = new LinkedList<string>();

linkedList.AddLast("Banana");

linkedList.AddLast("Date");

linkedList.AddLast("Fig");

linkedList.AddLast("Banana"); // Duplicates are allowed here.

Console.WriteLine("List 2 (Allows Duplicates - LinkedList):");

Console.WriteLine(string.Join(" -> ", linkedList));

Console.WriteLine("-----------------------------------");

// a. Perform Union

// The Union() method returns the unique elements from both collections.

var unionResult = uniqueList.Union(linkedList);

Console.WriteLine("\na. Union of the two lists:");

Console.WriteLine(string.Join(" -> ", unionResult));

Console.WriteLine("-----------------------------------");

// b. Perform Intersection

// The Intersect() method returns the elements that are common to both collections.

var intersectionResult = uniqueList.Intersect(linkedList);

Console.WriteLine("\nb. Intersection of the two lists:");

Console.WriteLine(string.Join(" -> ", intersectionResult));

Console.WriteLine("-----------------------------------");

// c. Combining corresponding elements if lists are of the same size

Console.WriteLine("\nc. Combining corresponding elements:");

// We use .Count for both HashSet and LinkedList to check their sizes.

if (uniqueList.Count == linkedList.Count)

{

// The Zip() method merges two sequences by combining elements at the same index.

var combinedList = uniqueList.Zip(linkedList, (first, second) => $"{first} & {second}");

Console.WriteLine("Combined List (since sizes are equal):");

Console.WriteLine(string.Join(" -> ", combinedList));

}

else

{

Console.WriteLine($"Lists cannot be combined as they are of different sizes ({uniqueList.Count} and {linkedList.Count}).");

}

Console.WriteLine("-----------------------------------");

// Example with lists of the same size to demonstrate combination

HashSet<string> sameSizeList1 = new HashSet<string> { "One", "Two", "Three" };

LinkedList<string> sameSizeList2 = new LinkedList<string>(new[] { "Four", "Five", "Six" });

Console.WriteLine("\nExample with lists of the same size:");

Console.WriteLine($"List A: {string.Join(" -> ", sameSizeList1)}");

Console.WriteLine($"List B: {string.Join(" -> ", sameSizeList2)}");

if (sameSizeList1.Count == sameSizeList2.Count)

{

var combinedList = sameSizeList1.Zip(sameSizeList2, (first, second) => $"{first} & {second}");

Console.WriteLine("Combined List:");

Console.WriteLine(string.Join(" -> ", combinedList));

}

}

}