# Write a program to convert input strings from lower to upper and upper to lower case.

Source Code:

using System;

namespace lab

{

internal class ConvertString

{

static void Main(string[] args)

{

Console.WriteLine("Enter a string to convert to uppercase:");

string inputUpper = Console.ReadLine();

string upperCase = inputUpper.ToUpper();

Console.WriteLine("Uppercase: " + upperCase);

Console.WriteLine("Enter a string to convert to lowercase:");

string inputLower = Console.ReadLine();

string lowerCase = inputLower.ToLower();

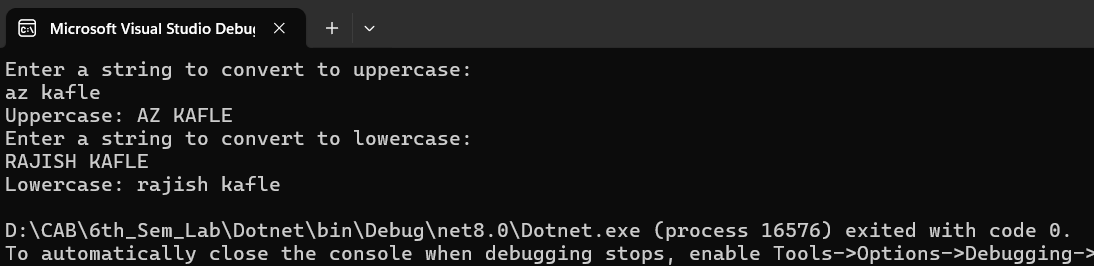
Console.WriteLine("Lowercase: " + lowerCase);

}

}

}

Output:



# Write a program to create a new string from a given string where first and last characters will be interchanged.

Source Code:

using System;

namespace lab

{

internal class StringInterchange

{

static void Main(string[] args)

{

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

string input = Console.ReadLine();

string result = SwapFirstAndLastCharacters(input);

Console.WriteLine("Modified string: " + result);

}

static string SwapFirstAndLastCharacters(string input)

{

if (string.IsNullOrEmpty(input) || input.Length == 1) return input;

char[] charArray = input.ToCharArray();

char firstChar = charArray[0];

char lastChar = charArray[input.Length - 1];

charArray[0] = lastChar;

charArray[input.Length - 1] = firstChar;

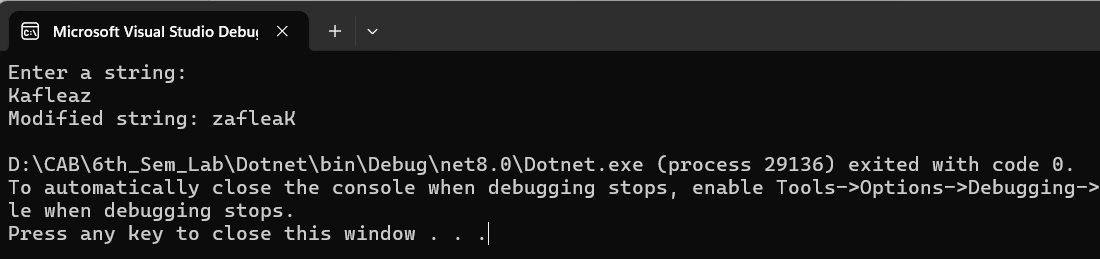
return new string(charArray);

}

}

}

Output:



# Write a program to demonstrate the basics of class and object.

Source Code:

using System;

namespace lab

{

internal class Basic\_class\_obj

{

class Person

{

public string? Name { get; set; }

public int Age { get; set; }

public void DisplayInfo()

{

Console.WriteLine($"Name: {Name}, Age: {Age}");

}

}

class Program

{

static void Main(string[] args)

{

Person person1 = new Person();

person1.Name = "AZ";

person1.Age = 23;

Person person2 = new Person();

person2.Name = "Rajish";

person2.Age = 24;

Console.WriteLine("Person 1:");

person1.DisplayInfo();

Console.WriteLine("\nPerson 2:");

person2.DisplayInfo();

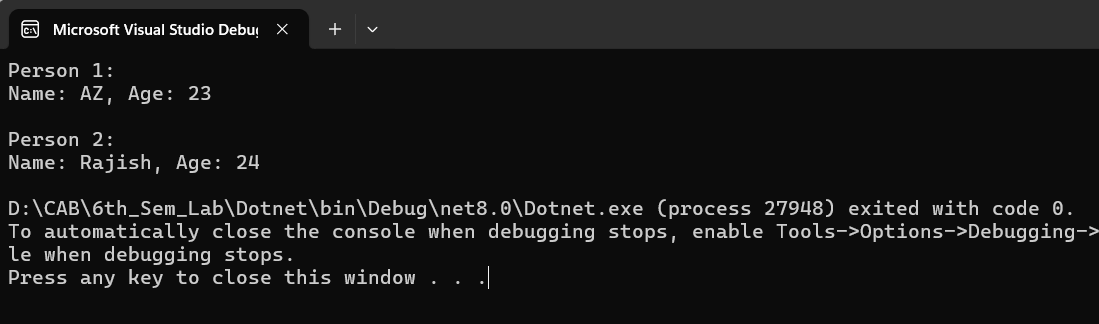
}

}

}

}

Output:



# Write a program to illustrate encapsulation with properties and indexers.

Source Code:

using System;

namespace lab

{

internal class Encapsulation

{

class Student

{

private string[] subjects = new string[5];

public string this[int index]

{

get { return subjects[index]; }

set { subjects[index] = value; }

}

public int TotalSubjects

{

get { return subjects.Length; }

}

}

class Program

{

static void Main(string[] args)

{

Student student = new Student();

student[0] = "Math";

student[1] = "Science";

student[2] = "History";

student[3] = "English";

student[4] = "Computer Science";

Console.WriteLine("Subjects:");

for (int i = 0; i < student.TotalSubjects; i++)

{

Console.WriteLine($"Subject {i + 1}: {student[i]}");

}

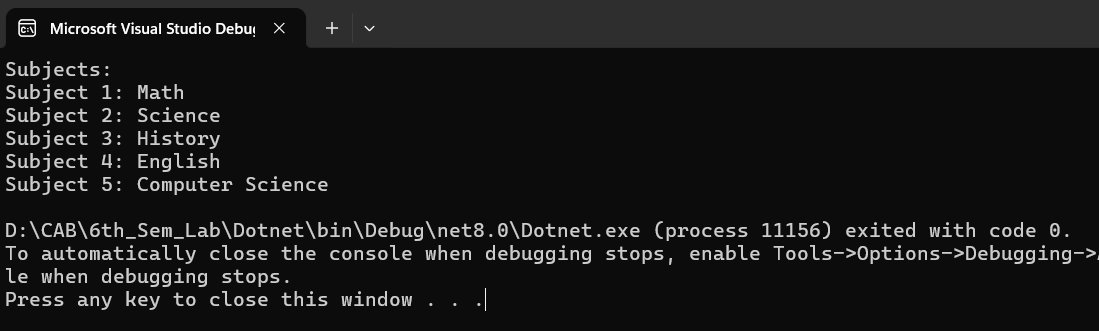
}

}

}

}

Output:



# Write a program that reflects the overloading and overriding of constructor and function.

Source Code:

using System;

namespace lab

{

internal class Overloading\_Riding

{

class Shape

{

public string Name { get; }

public Shape(string name)

{

Name = name;

}

public virtual void Display()

{

Console.WriteLine($"This is a {Name}");

}

}

class Rectangle : Shape

{

public double Width { get; }

public double Height { get; }

public Rectangle(string name, double width, double height) : base(name)

{

Width = width;

Height = height;

}

public override void Display()

{

base.Display();

Console.WriteLine($"It has width: {Width} and height: {Height}");

}

}

class Circle : Shape

{

public double Radius { get; }

public Circle(string name, double radius) : base(name)

{

Radius = radius;

}

public void Display(double area)

{

Console.WriteLine($"This is a {Name} with radius {Radius}");

Console.WriteLine($"Area: {area}");

}

}

class Program

{

static void Main(string[] args)

{

Rectangle rectangle = new Rectangle("Rectangle", 5, 10); rectangle.Display();

Circle circle = new Circle("Circle", 7);

double circleArea = CalculateCircleArea(circle.Radius);

circle.Display(circleArea);

}

static double CalculateCircleArea(double radius)

{

return Math.PI \* Math.Pow(radius, 2);

}

}

}

}

Output:



# Write a program to implement multiple inheritance with the use of interfaces.

Source Code:

using System;

namespace lab

{

internal class Multiple\_Inheritance

{

interface IShape

{

double CalculateArea();

}

interface IColor

{

string GetColor();

}

class Circle : IShape, IColor

{

private double Radius { get; }

private string Color { get; }

public Circle(double radius, string color)

{

Radius = radius;

Color = color;

}

public double CalculateArea()

{

return Math.PI \* Math.Pow(Radius, 2);

}

public string GetColor()

{

return Color;

}

}

class Program

{

static void Main(string[] args)

{

Circle redCircle = new Circle(5, "Red");

double area = redCircle.CalculateArea();

string color = redCircle.GetColor();

Console.WriteLine($"Circle Area: {area}");

Console.WriteLine($"Circle Color: {color}");

}

}

}

}

Output:



# Write a program to show how to handle exception in C#.

Source Code:

using System;

namespace lab

{

internal class ExceptionHandle

{

static void Main(string[] args)

{

try

{

Console.WriteLine("Enter a number:"); int num = int.Parse(Console.ReadLine());

int result = 10 / num;

Console.WriteLine($"Result: {result}");

}

catch (FormatException)

{

Console.WriteLine("Invalid input. Please enter a valid number.");

}

catch (DivideByZeroException)

{

Console.WriteLine("Division by zero is not allowed.");

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

}

finally

{

Console.WriteLine("Program execution completed.");

}

}

}

}

Output:



# Write a program to demonstrate use of Delegate and Events.

Source Code:

using System;

namespace lab

{

internal class Use\_Delegeate\_Events

{

public delegate void EventHandler(string message); class Publisher

{

public event EventHandler Notify;

public void DoSomething()

{

Console.WriteLine("Loading. . . . . ");

Notify?.Invoke("Loaded Successfully.");

}

}

class Subscriber

{

public void Subscribe(Publisher publisher)

{

publisher.Notify += HandleEvent;

}

public void Unsubscribe(Publisher publisher)

{

publisher.Notify -= HandleEvent;

}

private void HandleEvent(string message)

{

Console.WriteLine($"Event handled: {message}");

}

}

class Program

{

static void Main(string[] args)

{

Publisher publisher = new Publisher();

Subscriber subscriber = new Subscriber();

subscriber.Subscribe(publisher);

publisher.DoSomething();

subscriber.Unsubscribe(publisher);

publisher.DoSomething();

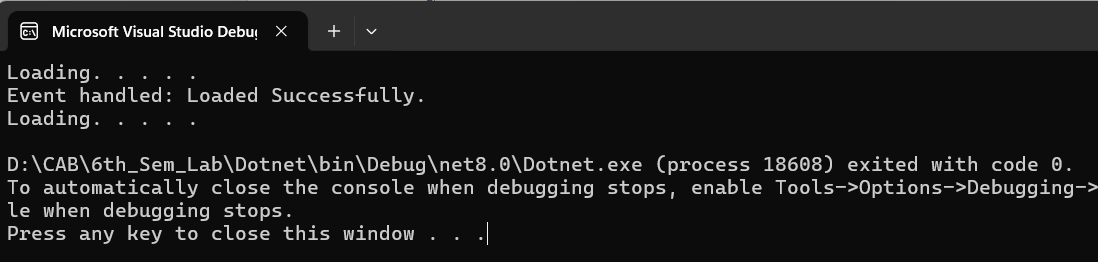
}

}

}

}

Output:



# Write a program to show the use of generic classes and methods.

Source Code:

using System;

namespace lab

{

internal class Generic\_Class

{

class Box<T>

{

private T contents; public Box(T item)

{

contents = item;

}

public T GetContents()

{

return contents;

}

}

class MathHelper

{

public static T Max<T>(T a, T b) where T : IComparable<T>

{

return a.CompareTo(b) > 0 ? a : b;

}

}

class Program

{

static void Main(string[] args)

{

Box<int> intBox = new

Box<int>(42); int intContents = intBox.GetContents();

Console.WriteLine($"Integer Contents: {intContents}");

Box<string> stringBox = new

Box<string>("Hello, Generics!");

string stringContents = stringBox.GetContents();

Console.WriteLine($"String Contents: {stringContents}");

int maxInt = MathHelper.Max(10, 20);

Console.WriteLine($"Max Integer: {maxInt}");

double maxDouble = MathHelper.Max(3.14, 2.71);

Console.WriteLine($"Max Double: {maxDouble}");

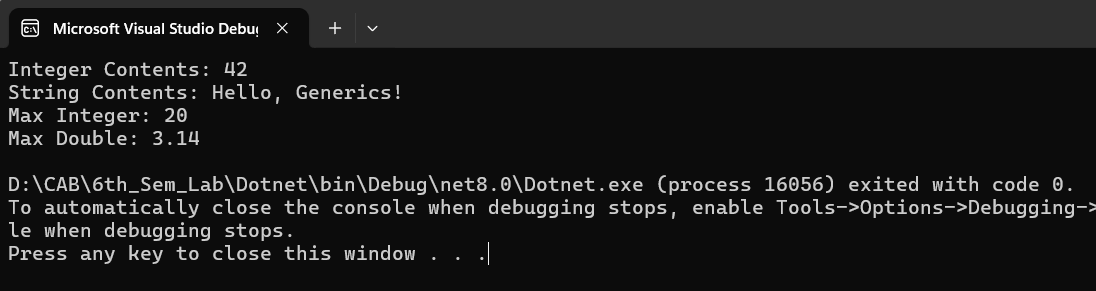
}

}

}

}

Output:



# Write a program to demonstrate the use of the method as a condition in the LINQ.

Source Code:

using System;

using System.Linq;

namespace lab

{

internal class Linq

{

class Person

{

public string Name { get; set; }

public int Age { get; set; }

}

class Program

{

static void Main(string[] args)

{

List<Person> people = new List<Person>

{

new Person { Name = "AZ", Age = 25 },

new Person { Name = "Rajish", Age = 30 },

new Person { Name = "Dev", Age = 22 }

};

var result = from person in people where IsAdult(person.Age) select person;

Console.WriteLine("Adults:");

foreach (var person in result)

{

Console.WriteLine($"Name: {person.Name}, Age: {person.Age}");

}

}

static bool IsAdult(int age)

{

return age >= 18;

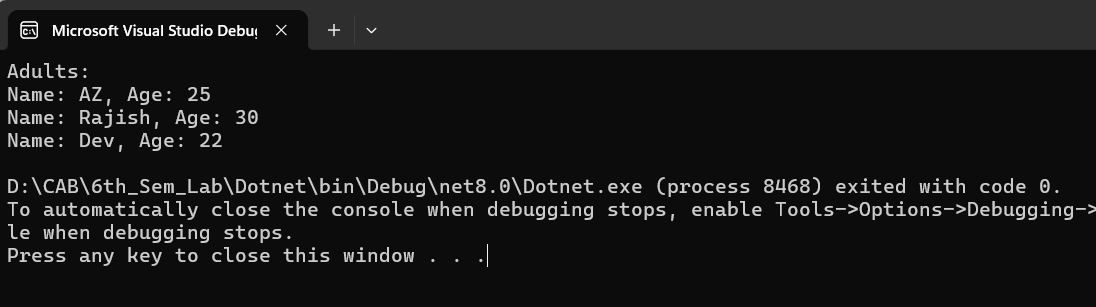
}

}

}

}

Output:



# Demonstrate Asynchronous programming with async, await, Task in C#.

Source Code:

using System;

namespace lab2

{

internal class Asynchronous

{

static void Main(string[] args)

{

Method1();

Method2();

Console.ReadKey();

}

public static async Task Method1()

{

await Task.Run(() =>

{

for (int i = 0; i < 10; i++)

{

Console.WriteLine(" Method 1");

// Do something

Task.Delay(100).Wait();

}

});

}

public static void Method2()

{

for (int i = 0; i < 5; i++)

{

Console.WriteLine(" Method 2");

// Do something

Task.Delay(100).Wait();

}

}

}

}

Output:

