Task 1 (5 points): Data types and control structures

Define an int variable called "temperature" and initialize it (e.g., 22). Define a string variable named "city" and initialize it with the name of your city. Write a loop that iterates from 1 to 7, and inside the loop print a message that includes the city, the current loop index, and the temperature in Fahrenheit. (Fahrenheit = (Celsius \* 9/5) + 32. ).

Assignment 2 (5 points): Classes and methods

Create a class named "Student" with the following properties: private integer field "\_studentName". Public string property "Grade" get and set with accessories. A constructor that takes two parameters to initialize Name and Grade. Create a public method called "Greeting" that returns the message: "Hi, I'm [\_studentName] and I'm in grade [Grade]"). Create an object of the "Student" class with the name and grade (Grade) and call the "Greeting" method.

Task 3 (5 points): Inheritance and method overriding

Create a class called "Figure" with the following: protected double property "Area." A constructor that initializes "Area" and assigns 0 to it. Create a child class called "Triangle" that inherits from "Shape" with the following: two private double fields "\_base" and "\_height" and properties corresponding to their names. Overload (Method Overloading) the "CalculateArea()" method in the "Triangle" class, which takes base and height parameters and calculates the area accordingly (formula: base \* height / 2). Create a child class called "Square" that also inherits from "Shape" with the following: private double field "\_sideLength and the property corresponding to the name." Override the "CalculateArea()" method in the "Square" class with a version that takes a square side parameter and calculates the area accordingly. Create objects of the "Triangle" and "Square" classes with different parameters (base, height or side of the square) Calculate and print the areas of both the triangle and the square using the overloaded "CalculateArea()" methods.

Task 4 (5 points): Abstract classes and interfaces

Create an abstract class "Animal" with the following members: the abstract method "MakeSound()" without implementing it. Create a descendant class of "Animal" class "Dog" that provides an implementation of "MakeSound()" text: "Woof." by returning Create a class "Cat", which is also a child of "Animal", and implements "MakeSound()" and returns the result "Meow". Create objects of the "Dog" and "Cat" classes and call their "MakeSound()" method.

Task 5 (5 points): Generics and general Generic List

Create a Generic class "MyCollection<T>". It should contain the following methods: Method "AddItem", which adds an element of type T to the List. Method "GetItem" that returns an item from the List according to the passed index. Create a "MyCollection" object with string type Generic and add some items to the list. Use the "GetItem" method to retrieve and print the item from the list based on the given index. Same with this

using System; // Importing the System namespace for basic functionalities

using System.Collections.Generic; // Importing the Collections.Generic namespace for using generic collections like List

// Main class that contains the entry point of the program

class Program

{

// The main method where execution starts

static void Main()

{

// Task 1: Data types and control structures

int temperature = 22; // Define an integer variable 'temperature' and initialize it with 22

string city = "YourCity"; // Define a string variable 'city' and initialize it with your city name

// Loop from 1 to 7 (inclusive)

for (int i = 1; i <= 7; i++)

{

// Calculate the temperature in Fahrenheit

double temperatureFahrenheit = (temperature \* 9.0 / 5.0) + 32;

// Print a message that includes the city, current loop index, and temperature in Fahrenheit

Console.WriteLine($"In {city}, day {i} has a temperature of {temperatureFahrenheit}°F.");

}

// Task 2: Classes and methods

// Create an instance of the Student class with name and grade

Student student = new Student("Alice", "A");

// Call the Greeting method and print the result

Console.WriteLine(student.Greeting());

// Task 3: Inheritance and method overriding

// Create an instance of Triangle and calculate its area

Triangle triangle = new Triangle();

double triangleArea = triangle.CalculateArea(5, 10); // Base 5, Height 10

Console.WriteLine($"Area of Triangle: {triangleArea}");

// Create an instance of Square and calculate its area

Square square = new Square();

double squareArea = square.CalculateArea(4); // Side length 4

Console.WriteLine($"Area of Square: {squareArea}");

// Task 4: Abstract classes and interfaces

// Create an instance of Dog and call its MakeSound method

Animal dog = new Dog();

Console.WriteLine(dog.MakeSound()); // Output: Woof

// Create an instance of Cat and call its MakeSound method

Animal cat = new Cat();

Console.WriteLine(cat.MakeSound()); // Output: Meow

// Task 5: Generics and general Generic List

// Create an instance of MyCollection for strings

MyCollection<string> myCollection = new MyCollection<string>();

myCollection.AddItem("Red"); // Add "Red" to the collection

myCollection.AddItem("Green"); // Add "Green" to the collection

myCollection.AddItem("Blue"); // Add "Blue" to the collection

// Retrieve and print the second item (index 1) from the collection

Console.WriteLine(myCollection.GetItem(1)); // Prints: Green

}

}

// Task 2: Classes and methods

public class Student

{

private string \_studentName; // Private field to store the student's name

public string Grade { get; set; } // Public property for grade with get and set accessors

// Constructor that takes name and grade as parameters

public Student(string studentName, string grade)

{

\_studentName = studentName; // Initialize the private student name field

Grade = grade; // Initialize the public Grade property

}

// Method that returns a greeting message

public string Greeting()

{

return $"Hi, I'm {\_studentName} and I'm in grade {Grade}."; // Return a formatted string

}

}

// Task 3: Inheritance and method overriding

public class Figure

{

protected double Area; // Protected field to store the area

// Constructor initializing Area to 0

public Figure()

{

Area = 0; // Set initial area to 0

}

}

// Triangle class that inherits from Figure

public class Triangle : Figure

{

private double \_base; // Private field for base

private double \_height; // Private field for height

// Overloaded method to calculate area based on base and height

public double CalculateArea(double baseLength, double height)

{

\_base = baseLength; // Assign base length

\_height = height; // Assign height

Area = (\_base \* \_height) / 2; // Calculate area using the formula (base \* height) / 2

return Area; // Return calculated area

}

}

// Square class that also inherits from Figure

public class Square : Figure

{

private double \_sideLength; // Private field for side length

// Property for SideLength

public double SideLength

{

get { return \_sideLength; } // Getter for SideLength

set { \_sideLength = value; } // Setter for SideLength

}

// Overridden method to calculate area based on side length

public double CalculateArea(double sideLength)

{

\_sideLength = sideLength; // Assign side length

Area = \_sideLength \* \_sideLength; // Calculate area using the formula side \* side

return Area; // Return calculated area

}

}

// Task 4: Abstract classes and interfaces

public abstract class Animal

{

public abstract string MakeSound(); // Abstract method to make a sound

}

// Dog class that inherits from Animal

public class Dog : Animal

{

public override string MakeSound()

{

return "Woof"; // Return the sound made by a Dog

}

}

// Cat class that also inherits from Animal

public class Cat : Animal

{

public override string MakeSound()

{

return "Meow"; // Return the sound made by a Cat

}

}

// Task 5: Generics and general Generic List

public class MyCollection<T>

{

private List<T> \_list = new List<T>(); // Private field to store the list of type T

// Method to add an item to the list

public void AddItem(T item)

{

\_list.Add(item); // Add the item to the list

}

**Explanation of the Structure**

**• Namespaces**: The code uses System for basic functionalities and System.Collections.Generic for working with lists.

**• Main Class and Method**: The entry point for the program where all tasks are executed sequentially.

**• Tasks**: Each task is defined with corresponding classes and methods, demonstrating concepts like classes, inheritance, abstract classes, and generics.

**• Comments**: Each line is annotated with comments to explain its purpose and functionality.

Feel free to ask if you have any questions or need further clarifications!

// Method to get an item from the list by index

public T GetItem(int index)

{

return \_list[index]; // Return the item at the specified index

}

}