**Experiment No 5:**

|  |
| --- |
| **Title:** Use of properties in any Application |
| **Aim:** To demonstrate how properties improve the effectiveness of software applications |

**Properties in C#: Encapsulation and Controlled Access**

In C#, properties provide a mechanism to encapsulate fields within a class and provide controlled access to them. They allow for the abstraction of data by exposing getters and setters, enabling better control over how data is accessed and modified.

**Definition and Syntax:**

In C#, properties are defined using a combination of get and set accessors within a class. The syntax for defining a property is as follows:

public <type> PropertyName

{

get { return <field>; }

set { <field> = value; }

}

Here, <type> represents the data type of the property, PropertyName is the name of the property, and <field> refers to the private field associated with the property.

**Encapsulation:**

Properties facilitate encapsulation by allowing us to hide the internal implementation details of a class while providing a public interface to interact with the data. By encapsulating fields within properties, we can control access to them and enforce validation rules or business logic.

**Getter and Setter:**

The get accessor is used to retrieve the value of the property, while the set accessor is used to assign a new value to the property. Getters and setters can include additional logic, such as data validation or error handling, before getting or setting the value of the associated field.

**Example:**

public class Person

{

private string name;

public string Name

{

get { return name; }

set { name = value; }

}

}

In this example, the Person class has a private field name encapsulated within the Name property. The get accessor returns the value of the name field, while the set accessor assigns a new value to the name field.

**Benefits:**

**Abstraction:** Properties abstract away the internal details of a class, providing a simplified interface for interacting with objects.

**Encapsulation:** Properties encapsulate fields, allowing for better control over access and modification of data.

**Validation:** Properties enable data validation and error handling, ensuring that the data remains consistent and valid.

Problem Statement: Managing Student Information with Properties

Develop a C# program to manage student information using properties. The program should allow users to create instances of a Student class, set and retrieve various attributes such as name, age, and grade using properties, and display the information.

using System;

public class Student

{

// Properties

public string Name { get; set; }

public int Age { get; set; }

public char Grade { get; set; }

// Constructors

public Student()

{

// Default constructor

}

public Student(string name, int age, char grade)

{

Name = name;

Age = age;

Grade = grade;

}

}

class Program

{

static void Main(string[] args)

{

// Create an instance of the Student class

Student student = new Student();

// Prompt user for student information

Console.Write("Enter student's name: ");

student.Name = Console.ReadLine();

Console.Write("Enter student's age: ");

student.Age = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter student's grade: ");

student.Grade = Convert.ToChar(Console.ReadLine());

// Display student information

Console.WriteLine("\nStudent Information:");

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

Console.WriteLine("Age: " + student.Age);

Console.WriteLine("Grade: " + student.Grade);

// Keep the console window open

Console.ReadLine();

}

}

**Conclusion:**

Properties are a fundamental concept in C# that allows for encapsulation, abstraction, and controlled access to data within classes. By defining properties, developers can create more maintainable and robust code that adheres to object-oriented principles.