Assignment_03

03 Object-Oriented Programming

Test your knowledge

1. What are the six combinations of access modifier keywords and what do they do?

- Private: Members declared as private are accessible only within the same class.
- Internal: Members declared as internal are accessible within the same assembly.
- Protected: Members declared as protected are accessible within the same class and its derived classes.
- Protected Internal: Members declared as protected internal are accessible within the same assembly or in a derived class.
- Private Protected: Members declared as private protected are accessible within the same class or in derived classes within the same assembly.

2.What is the difference between the static, const, and readonly keywords when applied to

a type member?

- Static: A static member is shared by all instances of the same type and can be accessed without creating an instance of the type.
- Const: A constant member has a value that is known at compile-time and cannot be changed after compilation.
- Readonly: A readonly member can be assigned a value only during construction of the object or within the declaration.

3. What does a constructor do?

A constructor is a special method in C# that is automatically executed when an object of a class is created. It is used to initialize the object's properties, fields, or state.

4. Why is the partial keyword useful?

1The partial keyword in C# is useful when you want to split a large class into smaller parts. You can use the partial keyword to create multiple parts of the same class in separate files, and they will be combined into a single class at compile-time.

5. What is a tuple?

A tuple is a lightweight data structure in C# that is used to group multiple values together into a single entity. It provides a convenient way to return multiple values from a method or pass multiple values to a method as a single argument.

6. What does the C# record keyword do?

The C# record keyword is a new feature introduced in C# 9.0 that makes it easier to define value types. A record is a value type that has automatically implemented properties and provides a convenient way to define immutable classes.

7. What does overloading and overriding mean?

Overloading means providing multiple methods with the same name but different parameters. Overriding means providing a new implementation for a method that is already defined in a base class.

8. What is the difference between a field and a property?

A field is a variable that represents the data of an object, whereas a property is a special kind of method that provides a way to access and manipulate the data of an object. Properties can have get and set accessors to read and write the value of the underlying field.

9. How do you make a method parameter optional?

To make a method parameter optional in C#, you can provide a default value in the method definition, and the caller can choose to pass in the argument or not.

10. What is an interface and how is it different from abstract class?

An interface is a blueprint of a class that defines a set of members (methods, properties, events, and indexers) that a class must implement. An interface cannot be instantiated on its own and is used to define a common contract between multiple classes. An abstract class, on the other hand, is a class that can have abstract methods (methods without implementation) and can be used as a base class for derived classes.

11. What accessibility level are members of an interface?

public

12. True/False. Polymorphism allows derived classes to provide different implementations

of the same method.

True

13. True/False. The override keyword is used to indicate that a method in a derived class is

providing its own implementation of a method.

True

14. True/False. The new keyword is used to indicate that a method in a derived class is providing its own implementation of a method.

True

15. True/False. Abstract methods can be used in a normal (non-abstract) class.

False

16. True/False. Normal (non-abstract) methods can be used in an abstract class.

True

17. True/False.Derived classes can override methods that were virtual in the base class.

True

18. True/False. Derived classes can override methods that were abstract in the base class.

True

19. True/False. In a derived class, you can override a method that was neither virtual non abstract in the base class.

False

20. True/False. A class that implements an interface does not have to provide an implementation for all of the members of the interface.

False

21. True/False. A class that implements an interface is allowed to have other members that aren't defined in the interface.

True

22. True/False. A class can have more than one base class.

True

23. True/False. A class can implement more than one interface.

True

Working with methods

1. Let’s make a program that uses methods to accomplish a task. Let’s take an array and reverse the contents of it. For example, if you have 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, it would become 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.

To accomplish this, you’ll create three methods: one to create the array, one to reverse the array, and one to print the array at the end.

Your Main method will look something like this:

```
static void Main(string[] args) {
int[] numbers = GenerateNumbers();
Reverse(numbers);
PrintNumbers(numbers);
}
```

The GenerateNumbersmethod should return an array of 10 numbers. (For bonus points, change the method to allow the desired length to be passed in, instead of just always being 10.)

The PrintNumbersmethod should use a foror foreachloop to print out each item in the array. The Reversemethod will be the hardest. Give it a try and see what you can make happen. If you get

stuck, here’s a couple of hints:

Hint #1:To swap two values, you will need to place the value of one variable in a temporary location to make the swap:

```
// Swapping a and b.
int a = 3;
int b = 5;
int temp = a;
a = b;
b = temp;
```

Hint #2:Getting the right indices to swap can be a challenge. Use a forloop, starting at 0 and going up to the length of the array / 2. The number you use in the forloop will be the index of the first number to swap, and the other one will be the length of the array minus the index minus 1. This is to account for the fact that the array is 0-based. So basically, you’ll be swapping array[index]with array[arrayLength – index – 1].

```
using System;

namespace ReverseArray
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] numbers = GenerateNumbers();
            Reverse(numbers);
            PrintNumbers(numbers);
        }

        static int[] GenerateNumbers()
        {
            int[] numbers = new int[10];
            for (int i = 0; i < numbers.Length; i++)
            {
                numbers[i] = i + 1;
            }
            return numbers;
        }

        static void Reverse(int[] numbers)
        {
            int length = numbers.Length;
            for (int i = 0; i < length / 2; i++)
            {
                int temp = numbers[i];
                numbers[i] = numbers[length - i - 1];
                numbers[length - i - 1] = temp;
            }
        }

        static void PrintNumbers(int[] numbers)
        {
            foreach (int number in numbers)
            {
                Console.Write(number + " ");
            }
            Console.WriteLine();
        }
    }
}
```

2. The Fibonacci sequence is a sequence of numbers where the first two numbers are 1 and 1, and every other number in the sequence after it is the sum of the two numbers before it. So the third number is 1 + 1, which is 2. The fourth number is the 2nd number plus the 3rd, which is 1 + 2. So the fourth number is 3. The 5th number is the 3rd number plus the 4th number: 2 + 3 = 5. This keeps going forever.

The first few numbers of the Fibonacci sequence are: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Because one number is defined by the numbers before it, this sets up a perfect opportunity for using recursion.

Your mission, should you choose to accept it, is to create a method called Fibonacci, which

takes in a number and returns that number of the Fibonacci sequence. So if someone calls Fibonacci(3), it would return the 3rd number in the Fibonacci sequence, which is 2. If someone calls Fibonacci(8), it would return 21.

In your Mainmethod, write code to loop through the first 10 numbers of the Fibonacci sequence and print them out.

Hint #1:Start with your base case. We know that if it is the 1st or 2nd number, the value will be 1.

Hint #2:For every other item, how is it defined in terms of the numbers before it? Can you come up with an equation or formula that calls the Fibonaccimethod again?

```
using System;

namespace FibonacciSequence
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 1; i <= 10; i++)
            {
                Console.WriteLine(Fibonacci(i));
            }
        }

        static int Fibonacci(int number)
        {
            if (number == 1 || number == 2)
            {
                return 1;
            }
            else
            {
                return Fibonacci(number - 1) + Fibonacci(number - 2);
            }
        }
    }
}
```

Designing and Building Classes using object-oriented principles

1. Write a program that that demonstrates use of four basic principles of object-oriented programming /Abstraction/, /Encapsulation/, /Inheritance/ and /Polymorphism/.

2. Use /Abstraction/ to define different classes for each person type such as Student and Instructor. These classes should have behavior for that type of person.

3. Use /Encapsulation/ to keep many details private in each class.

4. Use /Inheritance/ by leveraging the implementation already created in the Person class to save code in Student and Instructor classes.

5. Use /Polymorphism/ to create virtual methods that derived classes could override to create specific behavior such as salary calculations.

6. Make sure to create appropriate /interfaces/ such as ICourseService, IStudentService, IInstructorService, IDepartmentService, IPersonService, IPersonService (should have person specific methods). IStudentService, IInstructorService should inherit from IPersonService.

Person

Calculate Age of the Person

Calculate the Salary of the person, Use decimal for salary

Salary cannot be negative number

Can have multiple Addresses, should have method to get addresses

Instructor

Belongs to one Department and he can be Head of the Department

Instructor will have added bonus salary based on his experience, calculate his years of experience based on Join Date

Student

Can take multiple courses

Calculate student GPA based on grades for courses

Each course will have grade from A to F

Course

Will have list of enrolled students

Department

Will have one Instructor as head

Will have Budget for school year (start and end Date Time)

Will offer list of courses

```
using System;
using System.Collections.Generic;

interface IPersonService
{
    int CalculateAge();
    decimal CalculateSalary();
    List<Address> GetAddresses();
}
```

```
interface IStudentService : IPersonService
{
    void EnrollInCourse(Course course);
    double CalculateGPA();
}

interface IInstructorService : IPersonService
{
    void BelongsToDepartment(Department department);
    bool IsHeadOfDepartment();
    decimal CalculateExperience();
}

interface IDepartmentService
{
    void OfferCourses(List<Course> courses);
    void SetHead(Instructor instructor);
}

class Address
{
    public string Street { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string ZipCode { get; set; }
}

class Person : IPersonService
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public decimal Salary { get; set; }
    public List<Address> Addresses { get; set; }

    public int CalculateAge()
    {
        int age = DateTime.Today.Year - DateOfBirth.Year;
        if (DateTime.Today < DateOfBirth.AddYears(age))
            age--;

        return age;
    }

    public decimal CalculateSalary()
    {
        if (Salary < 0)
            throw new Exception("Salary cannot be negative");

        return Salary;
    }

    public List<Address> GetAddresses()
    {
        return Addresses;
    }
}

class Instructor : Person, IInstructorService
{
    public Department Department { get; set; }
    public DateTime JoinDate { get; set; }
    public bool IsDepartmentHead { get; set; }

    public void BelongsToDepartment(Department department)
    {
        Department = department;
    }
```

```
    public bool IsHeadOfDepartment()
    {
        return IsDepartmentHead;
    }

    public decimal CalculateExperience()
    {
        int years = DateTime.Today.Year - JoinDate.Year;
        decimal experience = years * 1000;
        return experience;
    }

    public override decimal CalculateSalary()
    {
        decimal baseSalary = base.CalculateSalary();
        decimal bonus = CalculateExperience();
        return baseSalary + bonus;
    }
}

class Student : Person, IStudentService
{
    public List<Course> Courses { get; set; }

    public void EnrollInCourse(Course course)
    {
        Courses.Add(course);
        course.EnrollStudent(this);
    }

    public double CalculateGPA()
    {
        double total = 0;
        int count = 0;
        foreach (var course in Courses)
        {
            total += (double)course.Grade;
            count++;
        }
        return total / count;
    }
}

class Course
{
    public string Name { get; set; }

class Department
{
    public Instructor Head { get; set; }
    public List<Course> Courses { get; set; }
    public DateTime BudgetStartDate { get; set; }
    public DateTime BudgetEndDate { get; set; }

    public Department(Instructor head, List<Course> courses, DateTime budgetStartDate, DateTime budgetEndDate)
    {
        Head = head;
        Courses = courses;
        BudgetStartDate = budgetStartDate;
        BudgetEndDate = budgetEndDate;
    }

    public decimal CalculateBudget()
    {
        // Implementation for calculating budget based on budget start and end date, courses, instructor salary, etc.
        return 0;
```

```
    }
}
```

7. Try creating the two classes below, and make a simple program to work with them, as described below

Create a Color class:

On a computer, colors are typically represented with a red, green, blue, and alpha (transparency) value, usually in the range of 0 to 255. Add these as instance variables.

A constructor that takes a red, green, blue, and alpha value.

A constructor that takes just red, green, and blue, while alpha defaults to 255 (opaque).

Methods to get and set the red, green, blue, and alpha values from a Colorinstance.

A method to get the grayscale value for the color, which is the average of the red, green and blue values.

Create a Ball class:

The Ball class should have instance variables for size and color (the Color class you just created). Let’s also add an instance variable that keeps track of the number of times it has been thrown.

Create any constructors you feel would be useful.

Create a Pop method, which changes the ball’s size to 0.

Create a Throw method that adds 1 to the throw count, but only if the ball hasn’t been popped (has a size of 0).

A method that returns the number of times the ball has been thrown.

Write some code in your Main method to create a few balls, throw them around a few times, pop a few, and try to throw them again, and print out the number of times that the balls have been thrown. (Popped balls shouldn’t have changed.)

```
using System;

class Color
{
    private int red, green, blue, alpha;

    public Color(int red, int green, int blue, int alpha)
    {
        this.red = red;
        this.green = green;
        this.blue = blue;
        this.alpha = alpha;
    }
```

```
    public Color(int red, int green, int blue) : this(red, green, blue, 255) { }

    public int Red { get { return red; } set { red = value; } }
    public int Green { get { return green; } set { green = value; } }
    public int Blue { get { return blue; } set { blue = value; } }
    public int Alpha { get { return alpha; } set { alpha = value; } }

    public int GrayscaleValue()
    {
        return (red + green + blue) / 3;
    }
}

class Ball
{
    private int size;
    private Color color;
    private int throwCount;

    public Ball(int size, Color color)
    {
        this.size = size;
        this.color = color;
        this.throwCount = 0;
    }

    public int Size { get { return size; } set { size = value; } }
    public Color Color { get { return color; } set { color = value; } }

    public void Pop()
    {
        size = 0;
    }

    public void Throw()
    {
        if (size > 0)
            throwCount++;
    }

    public int ThrowCount { get { return throwCount; } }
}

class Program
{
    static void Main(string[] args)
    {
        Ball ball1 = new Ball(10, new Color(255, 0, 0));
        Ball ball2 = new Ball(5, new Color(0, 255, 0));
        Ball ball3 = new Ball(20, new Color(0, 0, 255));

        ball1.Throw();
        ball1.Throw();
        ball2.Throw();
        ball3.Throw();

        ball2.Pop();

        ball3.Throw();
        ball3.Throw();

        Console.WriteLine("Ball 1 throw count: " + ball1.ThrowCount);
        Console.WriteLine("Ball 2 throw count: " + ball2.ThrowCount);
        Console.WriteLine("Ball 3 throw count: " + ball3.ThrowCount);
    }
}
```