03 OOP

- Test your knowledge

1. What are the six combinations of access modifier keywords and what do they do?
    - public: The type or member can be accessed by any other code in the same assembly or another assembly that references it. The accessibility level of public members of a type is controlled by the accessibility level of the type itself.
    - private: The type or member can be accessed only by code in the same `class` or `struct`.
    - protected: The type or member can be accessed only by code in the same `class`, or in a `class` that is derived from that `class`.
    - internal: The type or member can be accessed by any code in the same assembly, but not from another assembly.
    - protected internal: The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived `class` in another assembly.
    - private protected: The type or member can be accessed by types derived from the `class` that are declared within its containing assembly.

    https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers

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

   static: this member belong to the type itself, rather than any objects from the specific type

   const: the member is a constant

   readonly: we use readonly to modify a variable, and once we assign the value to the read-only member, the value cannot be changed

3. What does a constructor do?

   create an object of the class and initialize class members

4. Why is the partial keyword useful?

   It is possible to split the definition of a class, a struct, an interface or a method over two or more source files. Each source file contains a section of the type or method definition, and all parts are combined when the application is compiled.

   https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/partial-classes-and-methods

5. What is a tuple?

   Tuple is a data type that we can group multiple data elements together. One use case for tuple is to use tuple as return type so that we can return multiple values in one method.

   https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-tuples

6. What does the C# record keyword do?

   Record is a reference type in C#, and it's primarily intended for supporting immutable data models.

   https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record

7. What does overloading and overriding mean?

   overloading: compile-time polymorphism. It allows methods in the same class share the same name but different parameter lists.

   overridign: run-time polymorphism. Happens between base class and sub class, where subclass can have different implementation for the methods. These methods should share the same method signature, including access modifiers, return type, method name, and input parameters.

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

   Field stores the data of specific information that we want to keep for certain object, usually we should set fields as private.

   Property is a private data field plus get and set accessor, so that we can encapsulate our data.

9. How do you make a method parameter optional?

   assign the default value to it when implement the method

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

    interface is a collection of methods which are by default abstract and public and will be implemented by the derived class

    Abstract class vs. Interface:

    - abstract class provides a base class to its subclasses -- clear class hierachy
      interface defines common behaviros or functionalities that can be implemented by any class -- contract
    - one class can only inherit from one abstract class, but one class can implement multiple interfaces
    - Methods in abstract class can be abstract methods or non-abstract methods, but methods in interfaces are by default abstract and public

11. What accessibility level are members of an interface?

    by default 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.

    False. We should use override

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

    False. As long as we have an abstract method, the current class must be abstract

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.

    False. Derived classes MUST override the abstract methods from the base class.

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

    False. Only virtual or abstract methods can be override.

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. All the members of the interface must be implemented.

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.

    False. In C#, we only have single inheritance.

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

    True.

- Working with methods

    1. Generate int array, reverse it and print all numbers

```
public int[] GenerateNumbers(int n)
{
    int[] numbers = new int[n];
    Random r = new Random();
    for (int i = 0; i < numbers.Length; i++)
    {
        numbers[i] = r.Next(int.MinValue, int.MaxValue);
    }
    return numbers;
}

public void Reverse(int[] array)
{
    for (int i = 0, j = array.Length - 1; i < j; i++, j--)
    {
```

```
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }

    public void PrintNumbers(int[] numbers)
    {
        foreach (var num in numbers)
        {
            Console.Write(num + " ");
        }
    }
```

2. Fibonacci Sequence

```
public int Fibonacci(int i)
{
    if (i <= 2)
    {
        return 1;
    }
    int a = 1, b = 1, c = 0;
    for (int j = 3; j <= i; j++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}
```

3. OOP-related

    a. Person class

```
public abstract class Person
{
    public int Id { get; set; }
    private double salary;
    public double Salary
    {
        get { return salary; }
        set
        {
            if (salary < 0)
            {
                throw new ArgumentOutOfRangeException("Only positive values are allowed");
            }
```

```
            salary = value;
        }
    }
    public DateTime DateOfBirth { get; set; }
    public int Age
    {
        get { return DateTime.Now.Year - DateOfBirth.Year; }
    }
    public List<string> Address { get; set; }
    public void GetAddress(List<string> Address)
    {
        foreach (var item in Address)
        {
            Console.WriteLine(item);
        }
    }

}
```

b. Instructor class

```
public class Instructor : Person
{
    public int DepartmentId { get; set; }
    public Department Department { get; set; }

}
```

c. Student class

```
public class Student : Person
{
    public List<Course> SelectedCourses { get; set; }
}
```

d. Course class

```
public class Course
{
    public int Id { get; set; }
    public string Name { get; set; }
    public List<Student> EnrolledStudents { get; set; }
}
```

e. CourseEnrollment class

```
public enum Grade
{
    A,
    B,
    C,
    D,
    E,
    F
}
public class CourseEnrollment
{
    public int StudentId { get; set; }
    public int CourseId { get; set; }
    public Student Student { get; set; }
    public Course Course { get; set; }
    public Grade Grade { get; set; }
    public double CalculateGPA()
    {
        throw new NotImplementedException();
    }

}
```

f. Department class

```
public class Department
{
    public int Id { get; set; }
    public string DepartmentName { get; set; }
    public DateTime JoinDate { get; set; }
    public int YearOfExperience
    { get { return DateTime.Now.Year - JoinDate.Year; } }
    public decimal Budget { get; set; }
    public int HeadId { get; set; }
    public Instructor Head { get; set; }
    public List<Course> ProvidedCourses { get; set; }
}
```

4. Color and Ball

a. Color class

```
public class Color
    {
        public byte Red { get; set; }
        public byte Green { get; set; }
        public byte Blue { get; set; }
        public byte Alpha { get; set; }
```

```
        public Color(byte red, byte green, byte blue, byte alpha)
        {
            Red = red;
            Green = green;
            Blue = blue;
            Alpha = alpha;
        }
        public Color(byte red, byte green, byte blue)
        {
            Red = red;
            Green = green;
            Blue = blue;
            Alpha = 255;
        }
        public byte GetGrayscale(Color c)
        {
            return Convert.ToByte((c.Red + c.Green + c.Blue) / 3);
        }
    }
```

b. Ball class

```
public class Ball
{
    public int Size {get; set; }
    public Color Color { get; set; }
    public int ThrownTimes { get; set; }
    public void Pop()
    {
        Size = 0;
    }
    public void Throw()
    {
        if(Size != 0)
        {
            ThrownTimes++;
        }
    }
    public int GetTimes()
    {
        return ThrownTimes;
    }
}
```