You can find this on my github: <https://github.com/princesscorn/assignments-antra.git>

# 03 Object-Oriented Programming

## Test your knowledge

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

=>

|  |  |
| --- | --- |
| **Access modifiers** | **Access** |
| Public | access anywhere |
| Private | Access in current class |
| Protected | Access in current class and derived class |
| Internal | Access in current assembly |
| Private protected | Access in derived classes in the same assembly |
| Protected internal | Current assembly and derived class in another assembly |

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

=>

Static:

Static members can only be accessed by the static method

Static members can be accessed by ClassName.StaticMemberName

Const:

By default, a const is a static.

Const members can be accessed by ClassName.ConstMemberName.

Readonly:

Readonly member can be accessed by object, not classname.

### 3.What does a constructor do?

=> constructor is used to create an instance of a class and initialize the class fields.

4.Why is the partial keyword useful?

=>

There are several situations when splitting a class definition is desirable:

* When working on large projects, spreading a class over separate files enables multiple programmers to work on it at the same time.
* When working with automatically generated source, code can be added to the class without having to recreate the source file. Visual Studio uses this approach when it creates Windows Forms, Web service wrapper code, and so on. You can create code that uses these classes without having to modify the file created by Visual Studio.
* When using [source generators](https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/source-generators-overview) to generate additional functionality in a class.

### 5.What is a tuple?

=>

A tuple is a data structure that contains a sequence of elements of different data types.

### 6.What does the C# record keyword do?

=>

The record keyword to define a reference type that provides built-in functionality for encapsulating data.

### 7.What does overloading and overriding mean?

=> overloading means have same methods name, different parameters, or different return type in the same class.

Overriding means have same methods signature between base class and derived class.

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

=> Fields are normal variable members of a class. Generally, you should declare your fields as private, then use Properties to get and set their values.

Property is actually special method called “accessors”. Properties are called accessors because they offer a way to get and set a field if you have a private field. They have two codes inside; set{}; and get{}; called “property accessors”.

### 9.How do you make a method parameter optional?

=> Optional parameters are defined at the end of the parameter list, after any required parameters.

Each optional parameter has a default value as part of its definition. If no argument is sent for that parameter, the default value is used.

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

=> Interface is a contract, is a completely abstract class, can only contain abstract method and property.

Difference:

1) Interface supports multiple inheritance, abstract class supports single inheritance.

2) Interface cannot have constructor, but abstract class can.

3) Interface has by default all members abstract and public, but abstract class can have private members.

### 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.

=>False

### 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 GenerateNumbers method 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.)

ThePrintNumbers method should use a for or foreach loop to print out each item in the array. The Reverse method 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 for loop, starting at 0 and going up to the length of the array / 2. The number you use in the

For loop 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 UnderstandingTypes

{

class Program

{

public static void Main(string[] args)

{

int[] numbers = GenerateNumbers();

Reverse(numbers);

PrintNumbers(numbers);

}

public static int[] GenerateNumbers()

{

Console.WriteLine("Input how many numbers you want: ");

int totalNumbers = Convert.ToInt32(Console.ReadLine());

int[] numbers = new int[totalNumbers];

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

{

numbers[i] = i + 1;

}

PrintNumbers(numbers);

return numbers;

}

static void Reverse(int[] numbers)

{

int tmp = 0;

int totalNums = numbers.Length;

for (int i = 0; i < totalNums/2; ++i)

{

tmp = numbers[i];

numbers[i] = numbers[totalNums - i - 1];

numbers[totalNums - i - 1] = tmp;

}

}

static void PrintNumbers(int[] numbers)

{

for (int i = 0; i < numbers.Length; ++i)

{

Console.Write(numbers[i] + " ");

}

Console.WriteLine();

}

}

}

Text

Description automatically generated

### 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 Main method, 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 Fibonacci method again?

=>

using System;

namespace UnderstandingTypes

{

class Program

{

public static void Main(string[] args)

{

Console.WriteLine("Input a positive number: ");

uint number = Convert.ToUInt32(Console.ReadLine());

long fib = Fibonacci(number);

Console.WriteLine(fib);

}

static long Fibonacci(uint number)

{

long a = 1;

long b = 1;

long sum = 1;

for (uint i = 3; i <= number; ++i)

{

sum = a + b;

a = b;

b = sum;

}

return sum;

}

}

}

Text

Description automatically generated

## 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/.

namespace Demo

{

class Program

{

public static void Main()

{

FullTimeEmployee fte = new FullTimeEmployee();

//fte.AddFullTimeEmployee();

//fte.PrintFullTimeEmployee();

fte.LogInformation();

Employee fe = new FullTimeEmployee();

fe.LogInformation();

}

}

public abstract class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public int DeptId { get; set; }

public string Mobile { get; set; }

public void AddEmployee()

{

Console.WriteLine("Please Enter Id: ");

Id = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Please Enter Name: ");

Name = Console.ReadLine();

Console.WriteLine("Please Enter Department Id: ");

DeptId = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Please Enter Mobile Number: ");

Mobile = Console.ReadLine();

}

public void PrintEmployee()

{

Console.WriteLine("Id = " + Id);

Console.WriteLine("Name = " + Name);

Console.WriteLine("DeptId = " + DeptId);

Console.WriteLine("Mobile Number = " + Mobile);

}

public virtual void LogInformation()

{

Console.WriteLine("This is base class Employee.");

}

}

public class FullTimeEmployee : Employee

{

public Decimal Salary { get; set; }

public String Benefits { get; set; }

public void AddFullTimeEmployee()

{

AddEmployee();

Console.WriteLine("Please Enter Salary: ");

Salary = Convert.ToDecimal(Console.ReadLine());

Console.WriteLine("Please Enter Benefits: ");

Benefits = Console.ReadLine();

}

public void PrintFullTimeEmployee()

{

PrintEmployee();

Console.WriteLine("Salary = " + this.Salary);

Console.WriteLine("Benefits = " + Benefits);

}

public override void LogInformation()

{

Console.WriteLine("This is derived class FullTimeEmployee.");

}

}

}

### 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.

=>similar to 1

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

=>similar to 1

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

=>similar to 1

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

=>similar to 1

### 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

### 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.)

## Explore following topics

Fields

Access modifiers

Enumeration types

Constructors

Methods

Properties

Inheritance

Interfaces

Polymorphism

# 04 Generics

## Test your knowledge

### 1. Describe the problem generics address.

=>code reusable

=>solve the boxing and unboxing problem.

### 2. How would you create a list of strings , using the generic List class?

=> List<string> flowers = new List<string>();

### 3. How many generic type parameters does the Dictionary class have?

=> 2

### 4. True/False. When a generic class has multiple type parameters, they must all match.

=>True.

### 5. What method is used to add items to a List object?

=>List.Add()

List.AddRange()

List.Append()

### 6. Name two methods that cause items to be removed from a List.

=>List.Remove()

List.RemoveAt()

List.RemoveAll()

List.RemoveRang()

### 7. How do you indicate that a class has a generic type parameter?

=> TypeName <T>

### 8. True/False. Generic classes can only have one generic type parameter.

=> False

### 9. True/False. Generic type constraints limit what can be used for the generic type.

=> True.

### 10. True/False. Constraints let you use the methods of the thing you are constraining to.

=> True

## Practice working with Generics

### 1.Create a custom Stack class MyStack<T> that can be used with any data type which has following methods

1.int Count()

2.T Pop()

3.Void Push()

=>

public class MyStack<T>

{

private Stack<T> \_stack;

public int Count()

{

return \_stack.Count;

}

public T Pop()

{

return \_stack.Pop();

}

public void Push(T val)

{

\_stack.Push(val);

}

}

or

public class MyStack<T>

{

private T[] \_stack;

private int \_count;

private int \_size;

public MyStack()

{

\_count = 0;

\_size = 100;

\_stack = new T[\_size];

}

public int Count()

{

Console.WriteLine(\_count);

return \_count;

}

public T Pop()

{

T result = default(T);

if (\_count > 0)

{

result = \_stack[--\_count];

}

return result;

}

public void Push(T t)

{

if (\_count >= \_size)

{

Console.WriteLine("No space.");

return;

}

\_stack[\_count++] = t;

return;

}

}

### 2. Create a Generic List data structure MyList<T> that can store any data type. Implement the following methods.

1.void Add (T element)

2.T Remove (int index)

3.bool Contains (T element)

4.void Clear ()

5.void InsertAt (T element, int index)

6.void DeleteAt (int index)

7.T Find (int index)

=>

public class MyList<T> : ICollection, IEnumerable

{

private ArrayList \_innerList = new ArrayList();

public T this[int index]

{

get

{

return (T)\_innerList[index];

}

}

public void Add(T val)

{

\_innerList.Add(val);

}

public void Remove(int index)

{

\_innerList.RemoveAt(index);

}

public bool Contains(T val)

{

return \_innerList.Contains(val);

}

public void InsertAt(T val, int index)

{

\_innerList.Insert(index, val);

}

public void DeleteAt(int index)

{

\_innerList.Remove(index);

}

public T Find(int index)

{

\_innerList.Find(index);

}

}

### 3. Implement a GenericRepository<T> class that implements IRepository<T> interface that will have common /CRUD/ operations so that it can work with any data source such as SQL Server, Oracle, In-Memory Data etc. Make sure you have a type constraint on T were it should be of reference type and can be of type Entity which has one property called Id. IRepository<T> should have following methods

1.void Add(T item)

2.void Remove(T item)

3.Void Save()

4.IEnumerable<T> GetAll()

5.T GetById(int id)

=>

public interface IRepository<T> where T : class

{

public int Id { get; set; }

void Add(T item);

void Remove(int id);

void Save();

IEnumerable<T> GetAll();

T GetById(int id);

}

public class GenericRepository<T> : IRepository<T>

{

List<Department> \_list;

public GenericRepository()

{

\_list = new List<Department>();

}

public void Add(T item)

{

\_list.Add(item);

}

public void Remove(int id)

{

//\_list.Remove(id);

T d = GetById(id);

if (d != null)

{

\_list.Remove(d);

}

}

void Save()

{

throw new NotImplementedException();

}

IEnumerable<T> GetAll()

{

return \_list;

}

T GetById(int id)

{

foreach(var item in \_list)

{

if (item.Id == id)

{

return item;

}

}

return null;

}

}

Explore following topics

Generics in .NET: [Generics in .NET | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/standard/generics/)

Generic classes and methods: [Generic classes and methods | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/generics)

Collections and Data Structures: [Collections and Data Structures | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/standard/collections/)

Commonly Used Collection Types: [Commonly Used Collection Types | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/standard/collections/commonly-used-collection-types)

When to Use Generic Collections: [When to Use Generic Collections | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/standard/collections/when-to-use-generic-collections)