|  |  |  |
| --- | --- | --- |
| ****C# Basics and Fundamentals**** | | |
| 1. **No.** | **Topics** | **Page** |
| 1 | What is C#? | 4 |
| 2 | Explain the main features of C# | 4 |
| 3 | What is the difference between value types and reference types in C#? | 4 |
| 4 | What is a nullable type in C#? | 4 |
| 5 | Explain the concept of boxing and unboxing. | 5 |
| 6 | What is the purpose of the var keyword in C# | 5 |
| 7 | What is the difference between const and readonly in C# | 5 |
| 8 | What are properties in C# | 6 |
| 9 | Explain the concept of indexers in C# | 6 |
| 10 | What is the purpose of the using statement in C# | 7 |
| 11 | Explain the concept of extension methods | 7 |
| 12 | Explain the static constructor | 7 |
| 13 | Dynamic keyword | 8 |
| 14 | Explain the difference between == and Equals() in C#. | 8 |
| 15 | What are delegates in C#? | 8 |
| 16 | What is anonymous method in C#? | 9 |
| 17 | Explain the concept of lambda expressions. | 9 |
| 18 | What is LINQ? | 10 |
| 19 | How does exception handling work in C#? | 10 |
| 20 | Explain the concept of async and await in C#. | 10 |
| 21 | What is the purpose of the lock statement in C#? | 10 |
| 22 | What is a Task in C#? | 11 |
| 23 | Explain the concept of yield in C#. | 11 |
| 24 | What are events in C# | 12 |
| 25 | What is the difference between throw and throw ex? | 13 |
|  |  |  |
| ****Object-Oriented Programming (OOP)**** | | |
| 1 | Explain 4 pillars of OOPS | 14 |
| 2 | What is inheritance | 15 |
| 3 | What is Polymorphism | 15 |
| 4 | Explain concept of abstraction | 14 |
| 5 | What is encapsulation | 14 |
| 6 | What is the difference between class and object | 16 |
| 7 | What is Constructor | 16 |
| 8 | What is Destructor | 17 |
| 9 | What is method hiding | 17 |
| 10 | Difference between override, abstract and virtual | 17 |
| 11 | Explain the concept of Interface | 18 |
| 12 | Explain the abstract class | 18 |
| 13 | What is sealed class | 18 |
| 14 | Explain the sealed method | 18 |
| 15 | What is record | 19 |
| 16 | What is struct | 19 |
| 17 | Explain covariance and contra variance | 20 |
| 18 | What is design Pattern | 22 |
| 19 | What is Singleton Pattern | 22 |
| 20 | What is Factory Pattern | 23 |
| 21 | What is dependency injection | 24 |
|  |  |  |
| ****Advanced C# concepts**** | | |
| 1 | What is reflection? | 25 |
| 2 | Explain attributes | 25 |
| 3 | What is static class | 26 |
| 4 | Difference between abstract class and interface | 26 |
| 5 | What is multi threading | 26 |
| 6 | What is thread pool | 27 |
| 7 | Explain Thread Synchronization | 27 |
| 8 | What is Mutex | 28 |
| 9 | What is Semaphore | 29 |
| 10 | Explain Volatile keyword | 29 |
| 11 | What is interlocked in C# | 30 |
| 12 | What is concurrent dictionary | 31 |
|  |  |  |
| ****Miscellaneous Software Engineering Concepts**** | | |
| 1 | What is the difference between synchronous and asynchronous programming | 34 |
| 2 | Explain the concept of multithreading and parallelism. | 35 |
| 3 | What are the different ways to handle concurrency in .NET? | 27 |
| 4 | Describe the Task Parallel Library (TPL) in .NET. | 27 |
| 5 | What is async/await in C#? How does it help with asynchronous programming? | 27 |
| 6 | What are the benefits of using dependency injection? | 28 |
| 7 | What are some common design anti-patterns? | 28 |
| 8 | How do you perform performance tuning in a .NET application? | 29 |
| 9 | What are the best practices for exception handling in C#? | 30 |
| 10 | Explain the concept of garbage collection in .NET. | 30 |
| 11 | What are value types and reference types in C#? | 31 |
| 12 | How do you manage memory in .NET applications? | 32 |
| 13 | What is the purpose of IDisposable interface in C#? | 30 |
| 14 | Describe the concept of immutability in programming. | 31 |
| 15 | What is the difference between imperative and declarative programming? | 32 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**What is C#?**

* C# is a modern, high level programming langugue developed by Microsoft.
* It’s a part of .Net ecosystem, which is free, open source, cross platform framework that supports wide range of application development

**Explain the main features of C#**

* Object Oriented Programming - (Encapsulation, Inheritance, Polymorphism and Abstraction)
* Type Safety - enforces type constrains, helps us to perform operation on the compatible type reducing the runtime error
* Garbage Collection - automatic garbage collection, automatically remove the unused object from the memory. Reduces the developer’s effort of memory management
* Rich Library Support
* Cross Platform Development

**What is the difference between value types and reference types in C#?**

* Value Types are stored in the stack memory whereas ref type are stored in head memory only it’s address is stored in stack memory
* Stack is used in pace of small and immutable data and where performance is priority
* Heap is used in place of handling complex object, to share data across multiple places and places where null need to be handled

|  |  |
| --- | --- |
| **Stack** | **Heap** |
| Generally Faster | Heap allocation takes more time |
| Cannot be null unless made nullable | Can be null |
| Copies the Value | Copies the reference |

**What is nullable type in C#?**

* In C# the compiler won’t allow you to assign null value to a variable.
* C# 2.0 provide a special feature to allow us null value to a variable that is nullable type but only for the value type
* C# 8.0 provided us the feature to assign null values for the reference types also

**Explain the concept of boxing and unboxing.**

* **Boxing -** converting a value type to reference type
* **Un-boxing -** Converting a reference type to value type
* Boxing and unboxing are expensive operation because boxing involves heap allocation and the unboxing involves casting and type checking
* Avoid boxing and unboxing in performance critical application

**Explain ‘var’ keyword**

* ‘var’ is used to declare the implicitly typed variable, this means the type is determined at the compile time based on the initial value assigned to it.
* Once assigned the type cannot change
* It’s mostly used for declaring a cumbersome type variable and while using a LINQ query

**What is difference between const and readonly**

* Both are used to define a constant fields that are unchangeable. But the difference is where the value is assigned.
* ‘const’ - is a compile time constant. Whose value is initialized during the initialization. And the values cannot be changes.
* const is used where the value is known before the compilation and never changes
* ‘readonly’ - is a runtime constant, the value of readonly can be assigned while declaring or can be assigned in the constructor
* Readonly is used where the value is known at the runtime after compilation

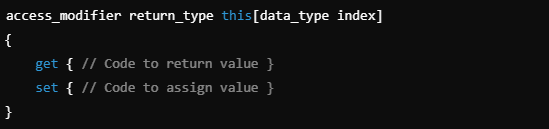
|  |  |
| --- | --- |
| **Const** | **readonly** |
| Value is assigned at declaration | Value is assigned at declaration or in constructor |
| Value cannot change | Value cannot change after assigned |
| Compile time field | Runtime field |

**What is properties in C#?**

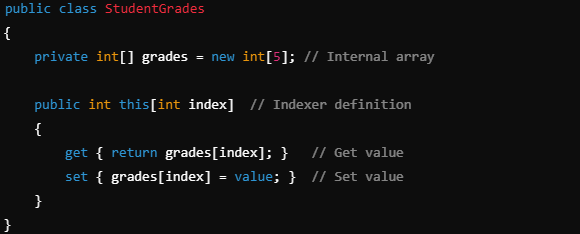
* Properties are special class member that is used to get and set the value of the field member safely, it allows controlled data access and modification
* Properties are used to where using for encapsulation (protected data access) , validataion and creating readonly or wirteonly variables.

**Explain the concept of Indexers?**

* Indexers are smart array the enables us to use the object of the class like array
* They provide a way to retrieve or set value of class or struct with the indexes

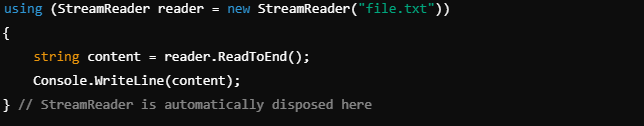


* Use ‘this’ keyword with the parameter inside []
* Use ‘get’ to retrieve value and ‘set’ to assign value



**Purpose of using keyword**

* ‘using’ can be used at two different context
* For managing the namespace
* For automatic resource management, it is used to release the resource of the object that implements the IDisposable Interface, by automatically calling the Dispose() method



**Explain the concept of Extension method**

* Extension method used to create additional functionality for the existing types without modifying the source code.
* They allow you to add methods to built-in types like string, int, List<T>, or even your custom classes.
* They are static methods but behave like instance methods of the type they extend.
* Syntax: they must be static and must be inside a static class and the first variable must use this before the type to extend

**Explain the concept of static constructor**

* Static constructor is a special constructor that is initialized only once when the constructor is accessed for the first time or when before any of the static property of member is used
* Static constructor cannot have access modifier and parameters and one class will contain only one static constructor

****

**Explain dynamic keyword**

* The dynamic keyword is used to define a variable whose type is determined at runtime
* It bypasses the compile time type checking, it can hold any type of data and can be changed at anytime and it reduce the explicit casting.
* We should not use dynamic type safety is necessary (because if we know the type then only, we can catch the error at compile time) and where the performance is critical

**Difference between == and Equals()**

* **‘==’** is used the compare the values but behaves different for the value type and reference type. For value type it checks the actual value and for the reference type it checks the memory address
* **‘.Equals()’** is the method exists in the object class, it will also behave same as the ‘==’ unless overridden for the reference type.

**What are delegates in C#?**

* Delegates are type safe function pointers in C#.
* It’s used to pass the method as a parameter, for callback function, for invoking multiple method and for event handling
* If a delegate is assigned with multiple methods which returns values, the delegate only returns the final value returned (the method which is subscribed at last)



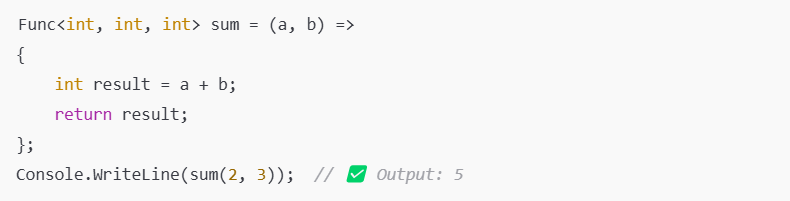
**What is anonymous method in C#?**

* Anonymous methods are method with name and is declared with the delegate keyword and is used to define inline method without explicitly defining them
* We can use anonymous method in place of callback, for short-lived method or when we want to reduce unnecessary method definition
* It can also access outer variables

****

**Explain the concept of lambda expression**

* Lambda expression is the shortest way to write the anonymous method in C#. We can declared a lambda expression without method name using the lambda operator ( => )
* It’s more concise and readable compared to anonymous function, no need to specify the delegate type explicitly and can have single and multiline expressions.
* **SYNTAX : (parameter) => (expression)**



**What is LINQ?**

* **LINQ -** stands for Language Integrated Query
* It is used to query Collection using SQL like syntax
* There are two syntax in LINQ, Query Syntax (sql like) and Method syntax

**How does exception handling work in C#?**

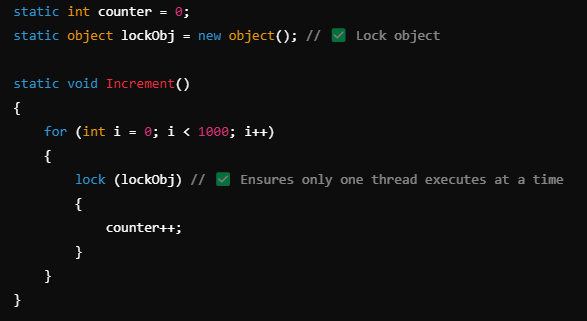
* Exception is a unhadled error occur at runtime, that stop the programs flow until it’s handled properly
* Exception handling is the mechanism that used to prevent the run time error and prevent the application from crashing down.
* Catches and manages the error in controlled manner

**Explain the concept of async and await**

* ‘async’ and ‘await’ are used for asynchronous programming
* They allow non blocking execution, which makes the application more responsive
* Await pause the execution until the Task completed, but not blocking the main thread, so the remaining process is not waiting
* ‘**async’** - keyword makes the method asynchronous
* ‘**await’ -** keyword is used to pause the execution of the program unit the Task completes, it won’t block the execution of the other processes

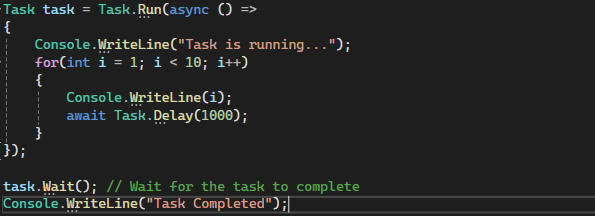
**What is the use of lock keyword?**

* ‘lock’ - keyword is used to lock a resource from being accessed by multiple tread at a same time, ensuring thread safety while using a shared resource
* If we try to access any shared resource without lock the output will be unpredictable (race - condition).



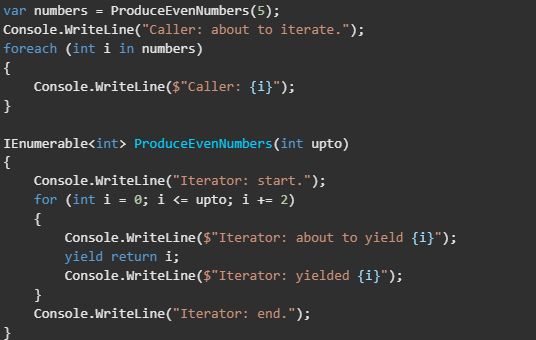
**What is Task?**

* A task is a asynchronous operation that runs in the back ground without blocking the main thread.
* It’s used for multi threading and asynchronous programming to improve applications performance.



**What is yield in C#?**

* Yield is used in iterator to return one value at a time this means not loading the entire collection in the memory.
* It helps in lazy evaluation that means values are generated only when needed.
* It is used to improve performance and efficient memory usage.



**Output:**

Caller: about to iterate.

Iterator: start.

Iterator: about to yield 0

Caller: 0

Iterator: yielded 0

Iterator: about to yield 2

Caller: 2

Iterator: yielded 2

Iterator: about to yield 4

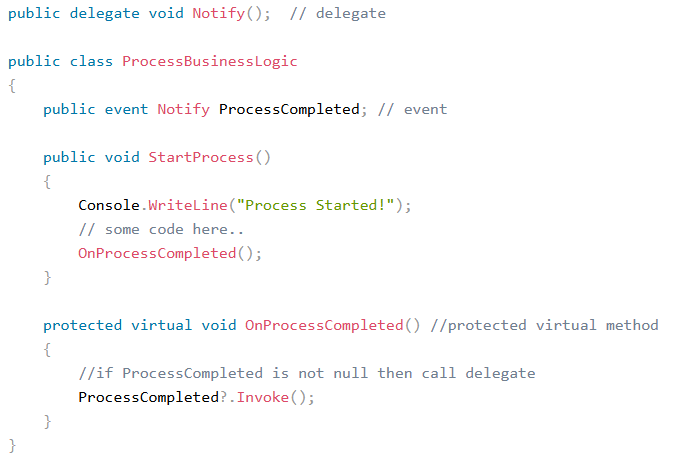
Caller: 4 Iterator: yielded 4

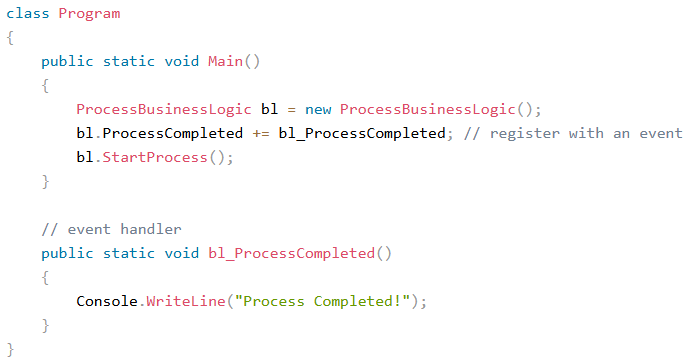
Iterator: end.

**What are events in C#?**

* Events are way for one object to notify another object when something happen. They are based on delegates and commonly used in the scenario like button click, data update and system notification.
* The class who raises event is called **Publisher** and the class who receives the notification is called **Subscriber.**
* There can be multiple subscriber for a single event.
* Typically, a publisher raises a event when some action occurs. The subscribers who are interested in getting notification when the event occurs may register with the event and handle it
* An event can be declared in two steps,

1. Declare a delegate.
2. Declare a variable of delegate with ‘event’ keyword.





**Throw vs throw ex**

* Both throw and throw ex used to rethrow the exception but the difference is the way they preserve the stack trace.
* ‘throw’ will preserve the original stack trace but the ‘throw ex’ will reset the stack trace the current method

### ****Object-Oriented Programming (OOP)****

****Explain 4 pillar of OOPS****

* **Encapsulation**
* **Absrtraction**
* **Inheritance**
* **Polymorphism**

****Encapsulation****

* **Encapsulation is process of hiding the internal details of the object and allowing access only through the controlled methods(getter and setter)**
* **It’s used to protect data from being modified accidentally and to provide security and maintainability**



**Abstraction**

* Abstraction is hiding the implementation and showing just only the necessary feature
* It can be achieve by using abstract classs or interface
* The user doesn’t need to know how something works, they just use it
* We will use abstraction when we enforces the rules( forces class to implement the methods

**Inheritance**

* Inheritance is one class inheriting from the other class
* The class which is being inherited is Parent/base class and the class which inherits is Child/Derived class
* The child class acquires all the public and protected members of class
* We should use inheritance to avoid code duplication, re-usability and when multiple classes shares the same functionality

**Polymorphism**

* Ability of an object to take multiple forms
* One method with different behaviour
* Two types

1. Method Overloading / Early Binding / Comiple time
2. Method Overriding / Late Binding / Runtime polymorphism

**Class and Object**

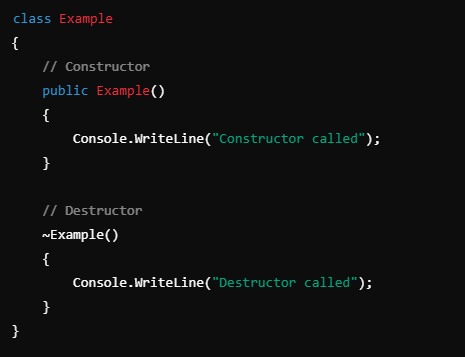
* Both the fundamental features of OOPS
* Class is skeleton that has properties and behaviours, for which object is being created
* Object is a real world entity that is created based on class that has the actual value

**What is constructor**

* Constructor is a special method that is being called when object for a class is created. Used to initialize the object’s property
* It should have same name as class, no return type
* Types,
  + Defalut Constructor (without Parameter)
  + Parameterized Constructor
  + Copy Constructor (Create new object by copying the existing object)
  + Static constructor
  + Private Constructor (Cannot Create Object , Used in singleton patter)

**What is Destructor?**

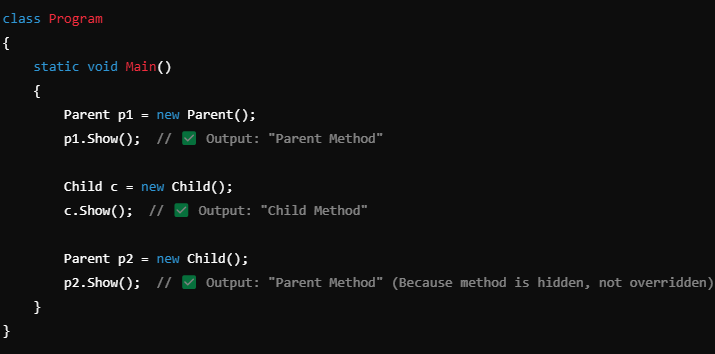
* Destructor is a special method that is being called automatically when the object is garbage collected
* It is used to clear the resources like files, database connections and the unmanaged memory before the object is destroyed.
* Syntax: **~ContructorName()** ( ~ tilde symbol with the constructor name)
* Cannot have parameter and return type, only one destructor per class
* Cannot call it explicitly, only invoked implictly when the object is garbage collected



**What is method hiding?**

* When you declare a method with the same name as in parent without the ‘override’ keyword then the child method hides the parent method.
* If you call the method with the parent class reference the parent method will be called even if it has the child class object
* If you call with the child class reference the child method will be called





**Difference between override, abstract and virtual**

* ‘**virtual’ -** declared a method that can be overridden in child
* ‘**override’ -** Overrides a virtual or abstract method in derived class
* ‘**abstract’ -** used to declare a method that must be implemented in derived class

**Explain the concept of interface**

* An interface is a buleprint that outlines the group of methods, events, indexers or properties that a class or struct must implement.
* The interface only have the definition of the methods, the implementation will be given by the class who implements the interface

**Explain the concept of abstract class**

* Abstract class cannot be instantiated and it’s meant to be inherited from some other classes.
* It can have both the concete and abstract method

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| Can have abstract and concrete method | Can have only method definition ( no implementation) |
| Can have fields | Cannot have fields |
| Can have constructors | Cannot have constructor |
| Can have access modifier | Always public by default |

**What is sealed class?**

* Sealed class cannot be inherited, it is used to prevent the further extension of a class ensuring no class can derive from it
* Used for security and performance - preventing inheritance can sometimes improves performance (as method calls are direct)

**Explain sealed method**

* Sealed method is used in inheritance to prevent the further overriding of a method in a derived class
* We can only use sealed on a overridden method in the derived class
* It is used to prevent the further modification of a method in sub classes and to enforce the behaviour of the method after overriding it once

**What is partial class**

* Partial class in C# allows you to split the definition of the class into multiple files
* This helps in organizing large classes, and making code manageable and improving teamwork in larger project

**What is record?**

* Record are special class in C# that is used for immutable, value based object.
* Unlike traditional classes which is reference type and are compared by reference, record are compared based on value equality
* By default the Property has ‘init’ so the record is immutable but we can changes that
* Records are used in Data Model, when two object is compared based on the value and for read-only objects where we won’t modify the object later

**What is struct**

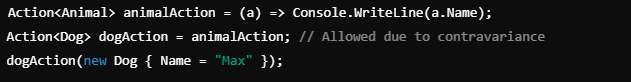
* Struct is a value type that is used to store small, lightweight object.
* It will store it’s value in the stack, which makes them faster for smaller data structure
* It’s sealed by default cannot be inherited
* It can’t be null (unless using nullable struct)

**What is Generics in C#?**

* C# generics allow you to declare class,interface, delegate and methods with the type parameter, they can work on any type without specifying the type beforehand
* Uses : Type safety : catch type errors at compile time rather that runtime, code re-usability and improve performance by avoiding boxing and unboxing for the value type
* Generics can also have constraints, that will force to have the specific type for the generic
* Eg: class Repository<T> where T : class // only reference type

**Explain covariance and contra variance**

* **Covariance** allows a type to replace a type by a more specific type this means you can use a child class where parent class is used, **Contravariance** allows a type to be replaces by a more general type this means you can use a parent class where derived class is used.
* Covariance use ‘out’ keyword. Contravariance using ‘in’
* Invariant means neither covariance npr contra variance
* Covariance is safe in output and contra variance is safe in input
* **IEnumerable<T> -->** T is covariant
* IEnumerable<Animal> animals2 =new List<Dog>();
* **Action** is contra variance

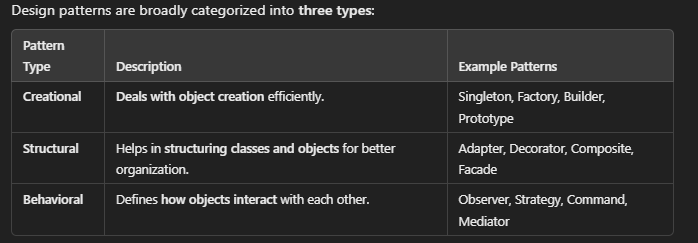


* Covariance cannot have a method that accept type parameter
* Contra variance cannot have method that return type parameter

|  |
| --- |
| using System;  namespace ContraVariance  {  public class CovarianceEg  {  public static void Main(string[] args)  {  // Covariance: Assigning a more derived type (Dog) to a base type (Animal)  Covariance<Dog> dog = new DogProvider();  Covariance<Animal> animal = dog;  Console.WriteLine(animal.GetAnimal());  // Contravariance: Assigning a base type (Animal) to a more derived type (Dog)  ContraVariance<Animal> animalC = new AnimalProcessor();  ContraVariance<Dog> dogC = animalC;  dogC.AcceptAnimal(new Dog { Name = "Labrador" });  }  }  // Base Class  public class Animal { }  // Derived Class  public class Dog : Animal  {  public string Name { get; set; }  }  // Covariant Interface (out: Can only return values)  interface Covariance<out T>  {  T GetAnimal();  // void SetAnimal(T animal); ❌ Not possible: Covariant type cannot accept type as a parameter.  }  // Contravariant Interface (in: Can only accept parameters)  interface ContraVariance<in T>  {  void AcceptAnimal(T animal);  // T GetAnimal(); ❌ Not possible: Contravariance cannot return a type parameter.  }  // Covariant Implementation: Provides Dog objects  class DogProvider : Covariance<Dog>  {  public Dog GetAnimal()  {  return new Dog { Name = "Labrador" };  }  }  // Contravariant Implementation: Processes any Animal object  class AnimalProcessor : ContraVariance<Animal>  {  public void AcceptAnimal(Animal animal)  {  Console.WriteLine($"Processing Animal...");  }  }  } |

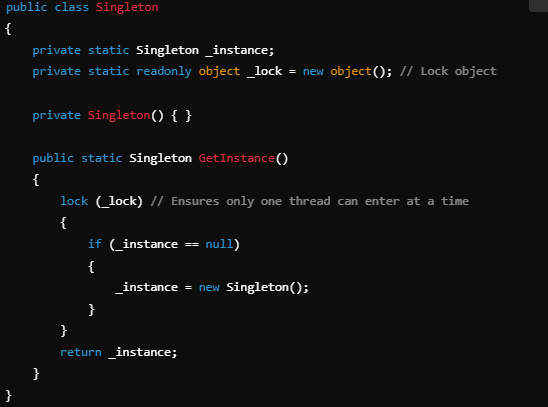
**What is design pattern?**

* Design pattern is a reusable solution to a commonly occurring software design problems.



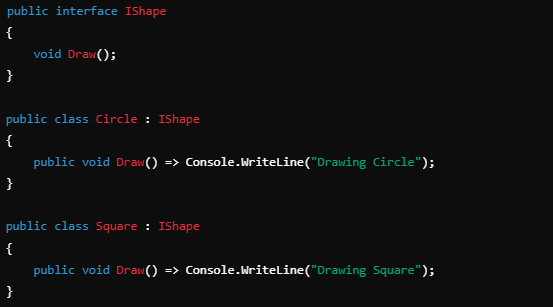
**What is singleton Pattern?**

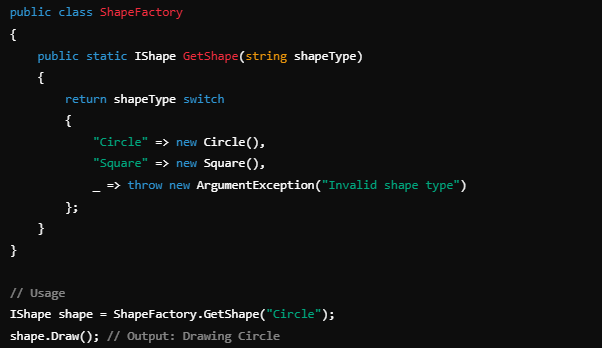
* It is a Creational design pattern to ensure that the class has only one object and provide global access point to that instance
* Useful for database connection, logging service, configuring settings
* Below is an example of thread safe singleton object creation, it is thread safe but little slower due to lock



**What is factory design Pattern?**

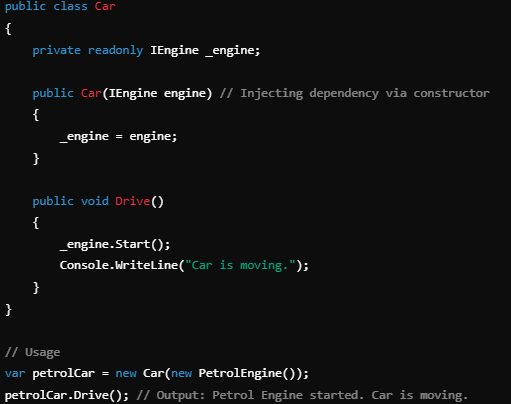
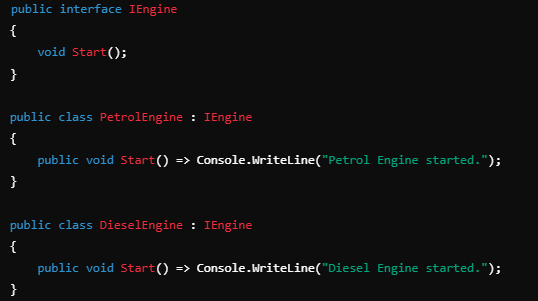
* It is a creational design pattern that is used to create a object of a class without specifying the exact class type. For object creation we will call our method created for creating object ( factory method) that will create and give object for us based on the condition
* It’s used to have a centralized place for object creation





**What is dependency injection?**

* Dependency injection is a design pattern used to achieve loose coupling between the object by injecting the dependency from outside rather than creating then inside.
* It’s is used to ease out the testing by creating the mock, allows loose coupling - reduce the dependency between classes, easy maintainability - change in one class doesn’t require modifying in the dependant class



* Asp .NET core provide inbuilt DI support.

**Advanced concept in C#**

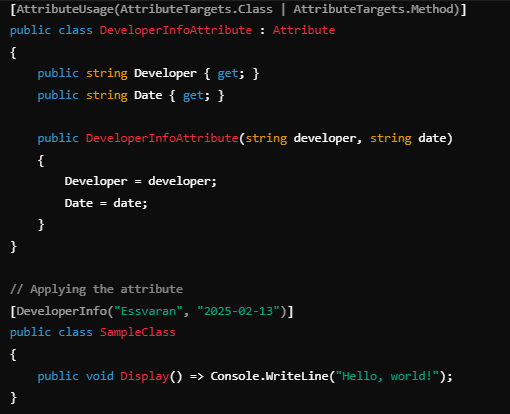
**What is reflection?**

* Reflection is the process of describing the metadata of the types, properties and classes.
* This namespace System.Reflection enables us to get the about the loaded assemblies, the elements within them like classes, methods and value types

<https://www.geeksforgeeks.org/what-is-reflection-in-c-sharp/>

**Explain the concept of attributes**

* Attribute in C# is a meta data in code that provides additional information about the classes, methods, properties or assemblies.
* The information can be accessed at runtime using the reflections
* Some built-in attributes [Obsolete] -- marks methods as outdated, [SuppressMessage] -- Controlling compiler warning
* Below is the example of custom attribute



**What is static class?**

* Static class is a class which cannot be instantiated, can only have static members( class, property and fields)
* Static class cannot be extended by any other class
* It is used to create utility/helper class that provide the common functionality without creating object.
* It’s sealed by default

**What is the difference between interface and abstract class?**

* An interface is a contract that has methods and properties without declaration, any child that implements the interface must provide the implementation for all of it’s members
* An abstract class is class that cannot be instantiated and may contain both abstract method and concrete method

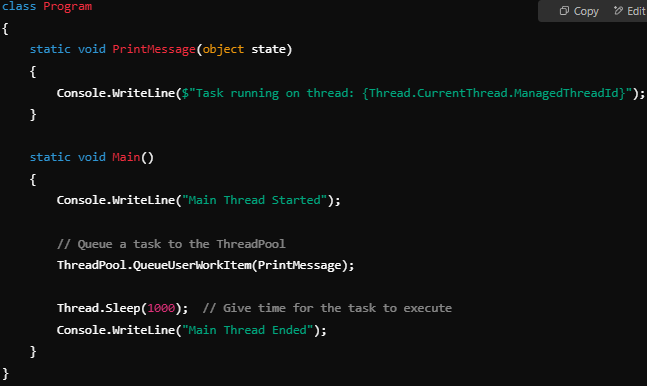
|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| Can have abstract and concrete method | Can have only method definition ( no implementation) |
| Can have fields | Cannot have fields |
| Can have constructors | Cannot have constructor |
| Can have access modifier | Always public by default |
| A class can only implement one abstract class | A class can implement multiple interfaces |

**What is multi threading?**

* Multithreading is a technique that runs multiple part of a application (process) at a same time, improving the performance and efficiency of the program.
* It helps to perform multiple task at the same time
* There are two types of threads
  + Foreground thread (keeps running until tasks complete even if the main thread stopped)
  + Background thread (stops when the main thread exits)
* By default all threads are foreground thread until setting them explicitly as background using **thread.IsBackGround = true**
* Multithreading may not be thread safe, may lead to **race condition (**when multiple thread try to access the shared resource at the same time), to handle this we use locking mechanism
* Multithreading improves the performance, maximize the CPU usage and multiple tasks at same time

**What is ThreadPool?**

* Thread pool is a collection of worker thread managed by .NET runtime that can be reused to perform background task effectively without creating the new Thread object
* Faster execution - avoid the delay of creating and destroying the thread



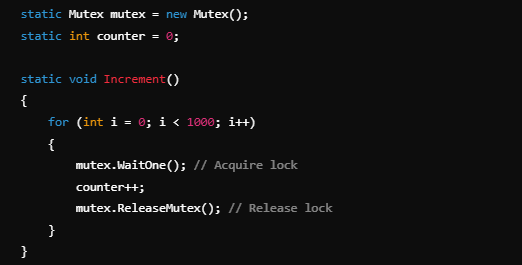
* **QueueUserWorkItem -** is used to assign a method to the thread pool thread, thread is reused from the thread pool instead of creating a new Thread.
* Thread pool is used for the short lived objects, when multiple short lived tasks need to run more effectively.
* When a task is used to run for a long time then we should use the Thread

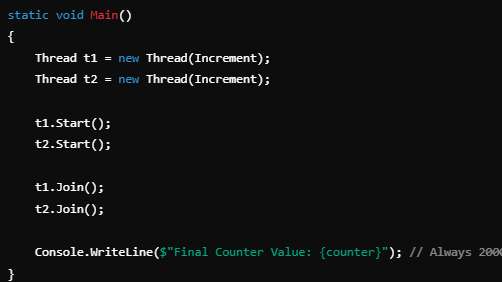
**Explain thread synchronization**

* Thread synchronization is a mechanism to control access to the shared resource in the multi threaded environment to prevent the race condition, deadlock or the inconsistent data.
* When multiple thread access the shared resource at same time conflict can occur.
* Synchronization ensures only one thread modifies the shared resource at a time, preventing the conflict
* Synchronization methods,
  + Lock keyword - Ensures only thread enters a code block at a time
  + Monitor class - Provide more control over thread synchronization
  + Mutex - Used for synchronization between processes
  + Semaphore - Allows a limited number of threads to access a resource
  + AutoResetEvent and MannualResetEvent - Used for signalling between threads

**What is mutex?**

* A mutex (Mutual Exclusion) is a synchronization object that prevent the multiple thread or processes from accessing the shared resource simultaneously.
* Unlike lock, mutex can work across multiple processes, making it useful for inter processes synchronization.
* We should manually release the mutex by **ReleaseMutex()** otherwise it will lead to deadlock
* For simple thread synchronization use lock (simple and faster)



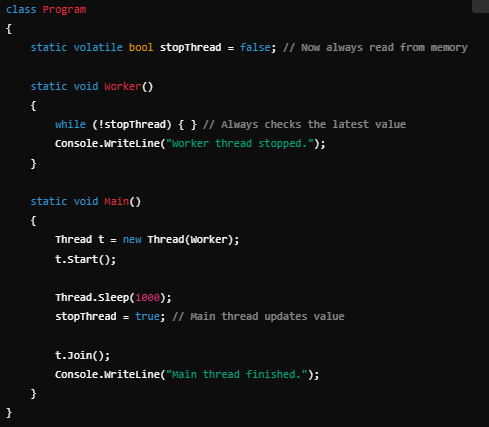


**What is semaphore?**

* Semaphore is a synchronization mechanism that limits the number is thread that can access the shared resource at a time
* Unlike thread and mutex which can only access one thread at a time, semaphore a access multiple thread but restricts the total numeber
* It’s useful in scenario where limited number of resources(data base connection, api calls, etc,..) need to be controlled.
* Always use Release() to avoid deadloack
* Two Types,
  + Semaphore (If multiple thread should access a resource, with a limit)
  + SemaphoreSlim (If only with a same process and you need performance)

**Explain volatile keyword**

* It is used to ensure that the value is read from the directly from the memory rather than being catched by memory
* This is used to prevent when multiple thread access and modifies the same value at a time.
* It ensures that always the latest value is fetched from the memory
* Use volatile when single thread writes and multi thread reads
* Don’t use volatile when multi thread reads and writes



**What is interlocked in C#**

* Interlock is a class that provide atomic operation (either the operation fully completes or not at all happen) for shared variable to prevent the race condition in the multi threaded environment
* Ensures that the operation is performed in a single indivisible step
* Prevents data corruption in multi threaded scenario, faster that the lock



**NOTES:**

* Use Interlock for simple counter/flag update
* Use thread for multiple related operation (ensures full block execution)
* Use volatile when only one thread writes and other thread reads (avoids catching issue).

**What is concurrent dictionary?**

* ConcurrentDictionary<TKey, TValue> is a thread safe dictionary that allow multiple thread to read and write without causing the race condition.
* It’s is part of System.Collections.Concurrent
* It’s thread safe and can be written without lock, more efficient than lock + dictionary

