Online Training
- C#.NET Online Training Program
- ASP.NET Core Training
- Microservices Online Training using .NET Core
- Microsoft Azure Training

Introduction & Environment Setup
- How Computer Works
- Introduction to Programming Languages
- How Computer Programs Works
- Different Types of Applications
- Programming Methodologies
- Algorithm, Pseudocode, Programs, and Flowcharts
- Introduction to .NET Framework
- .NET Framework Architecture and Components
- Introduction to C# Programming Language
- How to Download and Install Visual Studio on Windows
- Creating First Console Application using Visual Studio
- .NET Developer Roadmap for 2024
- Coding Standard Best Practices

C#.NET Basics
- Basic Structure of C# Program
- Methods and Properties of Console Class in C#
- Data Types in C#
- Literals in C#
- Type Casting in C#
- Variables in C#
- Operators in C#
- Control Flow Statements in C#
- If-Else Statements in C#
- Switch Statements in C#
- Loops in C#
- While Loop in C#
- Do While Loop in C#
- For Loop in C#
- Break Statement in C#
- Continue Statement in C#
- Goto Statement in C#
- Functions in C#
- User-Defined Functions in C#
- Call By Value and Call By Reference in C#
- Recursion in C#
- User Input and Output in C#
- Command Line Arguments in C#
- String in C#
- Static Keyword in C#
- Static vs Non-Static Members in C#
- Const and Read-Only in C#
- Properties in C#
- Why we Should Override ToString Method in C#

Differences Between Finalize and Dispose in C#

Back to: C#.NET Tutorials For Beginners and Professionals

Differences Between Finalize and Dispose in C#

In this article, I will explain the **Differences Between Finalize and Dispose in C#** with Example. Please read our previous article discussing **Garbage Collection in .NET Framework**. In C#, Finalize and Dispose are both methods used to release resources, but they serve different purposes and are used in different scenarios:

Finalize Method in C#:

- **Purpose:** The Finalize method is used for cleanup operations before an object is garbage collected. It's typically overridden to release unmanaged resources that the object holds. The garbage collector calls the Finalize method automatically.
- **Control:** You do not call Finalize directly. It's invoked by the garbage collector.
- **Non-deterministic:** The exact time when Finalize is called is non-deterministic, depending on the garbage collector's schedule.
- **Inheritance:** The Finalize method is inherited from the Object class. It should always call the Finalize method of its base class if overridden to ensure that all resources are released properly.

Dispose Method:

- **Purpose:** The Dispose method is part of the IDisposable interface and is implemented to release both managed and unmanaged resources deterministically.
- **Control:** Unlike Finalize, Dispose is called explicitly in your code, usually when you are done using an object. This allows for the immediate freeing of resources.
- **Deterministic:** Dispose provides a deterministic way to release resources, meaning you know exactly when the resources are released.
- **Pattern:** When implementing Dispose, it's common to follow the dispose pattern, which includes a finalizer call (GC.SuppressFinalize(this)) to prevent the garbage collector from calling Finalize if Dispose has already been called.

Example to Understand finalize and dispose in C#

Let us see an example to understand the use of Finalize and Dispose methods in resource management. In this example, we'll create a simple class named ResourceHolder that simulates the management of an unmanaged resource. The following example is self-explained, so please go through the comment line for better understanding.

```
using System;
namespace GarbageCollectionDemo
{
    public class ResourceHolder : IDisposable
    {
        // To track whether Dispose has been called.
        private bool _disposed = false;

        // Constructor
        public ResourceHolder()
        {
            // Allocate or initialize an unmanaged resource.
            Console.WriteLine("Unmanaged resource allocated.");
        }

        // Implementing Dispose method from IDisposable interface
        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this); // Prevent finalizer from being called.
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    // Free any other managed objects here.
                    Console.WriteLine("Free other managed objects");
                }
```

- Override Equals Method in C#
- Difference Between Convert.ToString and ToString Method in c#
- Checked and Unchecked Keywords in C#
- Stack and Heap Memory in .NET
- Boxing and Unboxing in C#

OOPs in C#

- Object Oriented Programming (OOPs) in C#
- Class and Objects in C#
- Constructors in C#
- Types of Constructors in C#
- Why We Need Constructors in C#
- Static vs Non-Static Constructors in C#
- Private Constructors in C#
- Destructors in C#
- Garbage Collection in .NET Framework
- **Differences Between Finalize and Dispose in C#**
- Access Specifiers in C#
- Encapsulation in C#
- Abstraction in C#
- Inheritance in C#
- Types of Inheritance in C#
- How to use Inheritance in Application Development
- IsA and HasA Relationship in C#
- Generalization and Specialization in C#
- Abstract Class and Abstract Methods in C#
- Abstract Class and Abstract Methods Interview Questions in C#
- How to Use Abstract Classes and Methods in C# Application
- Interface in C#
- Interface Interview Questions and Answers in C#
- Interface Realtime Examples in C#
- Multiple Inheritance in C#
- Multiple Inheritance Realtime Example in C#
- Polymorphism in C#
- Method Overloading in C#
- Operator Overloading in C#
- Method Overriding in C#
- Method Hiding in C#
- Partial Class and Partial Methods in C#
- Sealed Class and Sealed Methods in C#
- Extension Methods in C#
- Static Class in C#
- Variable Reference and Instance of a Class in C#

OOPs Real-Time Examples

- Real-time Examples of Encapsulation Principle in C#
- Real-Time Examples of Abstraction Principle in C#
- Real-Time Examples of Inheritance Principle in C#


```
                // Free unmanaged resources here.
                Console.WriteLine("Unmanaged resource released.");
                _disposed = true;
            }
        }

        // Finalizer is nothing but the destructor
        ~ResourceHolder()
        {
            Dispose(false);
            Console.WriteLine("Finalizer called.");
        }
    }

    //How to Use the above class
    class Program
    {
        static void Main(string[] args)
        {
            // Using the ResourceHolder with using statement
            using (var resourceHolder = new ResourceHolder())
            {
                // Use the resource...
            } // Dispose is called automatically when exiting the using block.

            // If not using 'using', dispose should be called manually.
            var anotherResourceHolder = new ResourceHolder();
            // Use the resource...
            anotherResourceHolder.Dispose();

            // Without calling Dispose, finalizer will be called by GC at some point.
            var finalResourceHolder = new ResourceHolder();
            // Use the resource...

            Console.ReadKey();
        }
    }
}
```

Output:

```
Unmanaged resource allocated.
Free other managed objects
Unmanaged resource released.
Unmanaged resource allocated.
Free other managed objects
Unmanaged resource released.
Unmanaged resource allocated.
```

Explanation:

ResourceHolder Class:

- The ResourceHolder class simulates the management of an unmanaged resource.
- The Dispose method is implemented from the IDisposable interface. It's called to release resources deterministically.
- The protected Dispose(bool disposing) method performs the actual resource cleanup. The disposing parameter indicates whether the method is being called from the Dispose method or from the finalizer.
- The finalizer (~ResourceHolder) is called by the garbage collector if the object is not disposed of properly. It calls Dispose(false).

Using the Class:

- The using statement ensures that Dispose is called automatically for resourceHolder.
- If not using using, Dispose should be called manually as shown with anotherResourceHolder.
- finalResourceHolder demonstrates a scenario where Dispose is not called, hence the finalizer will eventually be invoked by the garbage collector.

The example above illustrates the proper implementation and usage of both Dispose and Finalize for resource management in C#.

Key Differences Between Finalize and Dispose in C#:

- **Timing:** Finalize is called by the garbage collector in a non-deterministic manner, while Dispose is called explicitly at a known point in the program.
- **Resources:** Finalize is typically used for unmanaged resources, whereas Dispose can be used for both managed and unmanaged resources.
- **Control:** Dispose gives you more control over resource management compared to Finalize.

Best Practices:

Real-Time Examples of Polymorphism Principle in C#
Real-Time Examples of Interface in C#
Real-Time Examples of Abstract Class in C#

Exception Handling
Exception Handling in C#
Multiple Catch Blocks in C#
Finally Block in C#
How to Create Custom Exceptions in C#
Inner Exception in C#
Exception Handling Abuse in C#

Events, Delegates and Lambda Expression in C#
Course Structure of Events, Delegates and Lambda Expression
Roles of Events, Delegates and Event Handler in C#
Delegates in C#
Multicast Delegates in C#
Delegates Real-Time Example in C#
Generic Delegates in C#
Anonymous Method in C#
Lambda Expressions in C#
Events in C# with Examples

Multi-Threading
Multithreading in C#
Thread class in C#
How to Pass Data to Thread Function in Type Safe Manner in C#
How to Retrieve Data from a Thread Function in C#
Join Method and IsAlive Property of Thread Class in C#
Thread Synchronization in C#
Lock in C#
Monitor Class in C#
Mutex Class in C#
Semaphore Class in C#
SemaphoreSlim Class in C#
Deadlock in C#
Performance Testing of a Multithreaded Application
Thread Pool in C#
Foreground and Background Threads in C#
AutoResetEvent and ManualResetEvent in C#
Thread Life Cycle in C#
Threads Priorities in C#
How to Terminate a Thread in C#
Inter Thread Communication in C#
How to Debug a Multi-threaded Application in C#

Collections in C#
Arrays in C#
2D Arrays in C#
Advantages and Disadvantages of Arrays in C#
Collections in C#
ArrayList in C#
Hashtable in C#
Non-Generic Stack in C#
Non-Generic Queue in C#
Non-Generic SortedList in C#

- Implement Dispose to allow deterministic cleanup of resources.
- Use a finalizer (Finalize method) only for cleaning up unmanaged resources that are not wrapped in a safe handle and when there's no guarantee that Dispose will be called.
- In the Dispose method, call GC.SuppressFinalize to prevent the garbage collector from calling Finalize, if it has already been disposed of.
- Follow the dispose pattern, especially if your class owns unmanaged resources.

In the next article, I will discuss **Access Specifiers in C#** with Examples. In this article, I explain the **Differences Between Finalize and Dispose in C#** with Example. I hope you enjoy this Differences Between Finalize and Dispose in C# with Example article.

Dot Net Tutorials
About the Author: Pranaya Rout

Pranaya Rout has published more than 3,000 articles in his 11-year career. Pranaya Rout has very good experience with Microsoft Technologies, Including C#, VB, ASP.NET MVC, ASP.NET Web API, EF, EF Core, ADO.NET, LINQ, SQL Server, MYSQL, Oracle, ASP.NET Core, Cloud Computing, Microservices, Design Patterns and still learning new technologies.

Previous Lesson
Garbage Collection in .NET Framework

Next Lesson
Access Specifiers in C#

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name* Email* Website

Post Comment

- Advantages and Disadvantages of Non-Generic Collection in C#
- Generic Collections in C#
- Generics in C#
- Generic Constraints in C#
- Generic List Collection in C#
- How to Sort a List of Complex Type in C#
- Comparison Delegate in C#
- Dictionary Collection Class in C#
- Conversion Between Array List and Dictionary in C#
- List vs Dictionary in C#
- Generic Stack Collection Class in C#
- Generic Queue Collection Class in C#
- Foreach Loop in C#
- Generic HashSet Collection Class in C#
- Generic SortedList Collection Class in C#
- Generic SortedSet Collection Class in C#
- Generic SortedDictionary Collection Class in C#
- Generic LinkedList Collection Class in C#
- Concurrent Collection in C#
- ConcurrentDictionary Collection Class in C#
- ConcurrentQueue Collection Class in C#
- ConcurrentStack Collection Class in C#
- ConcurrentBag Collection Class in C#
- BlockingCollection in C#

File Handling

- File Handling in C#
- FileStream Class in C#
- StreamReader and StreamWriter in C#
- File Class in C#
- TextWriter and TextReader in C#
- BinaryWriter and BinaryReader in C#
- StringWriter and StringReader in C#
- FileInfo Class in C#
- DirectoryInfo Class in C#
- Export and Import Excel Data in C#

Asynchronous Programming

- Introduction to Concurrency
- Async and Await in C#
- Task in C#
- How to Return a Value from Task in C#
- How to Execute Multiple Tasks in C#
- How to Limit Number of Concurrent Tasks in C#
- How to Cancel a Task in C# using Cancellation Token
- How to Create Synchronous Method using Task in C#
- Retry Pattern in C#
- Only One Pattern in C#
- How to Control the Result of a Task in C#

- Task-Based Asynchronous Programming in C#
- Chaining Tasks by Using Continuation Tasks
- How to Attached Child Tasks to a Parent Task in C#
- ValueTask in C#
- How to Cancel a Non-Cancellable Task in C#
- Asynchronous Streams in C#
- How to Cancel Asynchronous Stream in C#

Parallel Programming

- Task Parallel Library in C#
- Parallel For in C#
- Parallel Foreach Loop in C#
- Parallel Invoke in C#
- Maximum Degree of Parallelism in C#
- How to Cancel Parallel Operations in C#
- Atomic Methods Thread Safety and Race Conditions in C#
- Interlocked vs Lock in C#
- Parallel LINQ in C#
- Multithreading vs Asynchronous Programming vs Parallel Programming in C#

AutoMapper

- AutoMapper in C#
- AutoMapper Complex Mapping in C#
- How to Map Complex Type to Primitive Type using AutoMapper in C#
- AutoMapper Reverse Mapping in C#
- AutoMapper Conditional Mapping in C#
- AutoMapper Ignore Method in C#
- Fixed and Dynamic Values in Destination Property in AutoMapper

Optional Parameter, Indexers and Enums

- How to make Optional Parameters in C#
- Indexers in C#
- Indexers Real-Time Example in C#
- Enums in C#

.NET Framework Architecture

- DOT NET Framework
- Common Language Runtime in .NET Framework
- .NET Program Execution Process
- Intermediate Language (ILDASM & ILASM) Code in C#
- Common Type System in .NET Framework
- Common Language Specification in .NET Framework
- Managed and Unmanaged Code in .NET Framework
- Assembly DLL EXE in .NET Framework
- App Domain in .NET Framework
- Strong and Weak Assemblies in .NET Framework
- How to Install an Assembly into GAC in .NET Framework

DLL Hell Problem and Solution in .NET Framework

Var, Dynamic and Reflection

- Reflection in C#
- Dynamic Type in C#
- Var Keyword in C#
- Var vs Dynamic in C#
- Dynamic vs Reflection in C#
- Volatile Keyword in C#
- Ref vs Out in C#
- Named Parameters in C#

C# 7.X new Features

- C# 7 New Features
- Enhancement in Out Variables in C# 7
- Pattern Matching in C#
- Digit Separators in C# 7
- Tuples in C# 7
- Splitting Tuples in C# 7
- Local Functions in C# 7
- Ref Returns and Ref Locals in C# 7
- Generalized Async Return Types in C# 7
- Expression Bodied Members in C#
- Thrown Expression in C#
- Async Main in C#

C# 8 New Features

- C# 8 New Features
- ReadOnly Structs in C#
- Default Interface Methods in C#
- Pattern Matching in C#
- Using Declarations in C#
- Static Local Functions in C#
- Disposable Ref Structs in C#
- Nullable Reference Types in C# 8
- Asynchronous Streams in C#
- Asynchronous Disposable in C#
- Indices and Ranges in C#
- Null-Coalescing Assignment Operator in C#
- Unmanaged Constructed Types in C#
- Stackalloc in in C#

Most Popular C# Books

- Most Recommended C# Books
- Most Recommended Data Structure and Algorithms Books using C#

© Dot Net Tutorials | Website Design by Sunrise Pixel