

# 1 System.Exception Class

```
2 [ILAsm]  
3 .class public serializable Exception extends System.Object  
  
4 [C#]  
5 public class Exception
```

## 6 Assembly Info:

- 7 • *Name:* mscorlib
- 8 • *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- 9 • *Version:* 2.0.x.x
- 10 • *Attributes:*
  - 11 ○ CLSCompliantAttribute(true)

## 12 Summary

13 Represents errors that occur during application execution.

## 14 Inherits From: System.Object

15

16 **Library:** BCL

17

18 **Thread Safety:** All public static members of this type are safe for multithreaded operations.  
19 No instance members are guaranteed to be thread safe.

20

## 21 Description

22 This class is the base class for all Exceptions.

23

24 When an error occurs, either the system or the currently executing application reports it  
25 by throwing an Exception containing information about the error. Once thrown, an  
26 Exception is handled by the application or by the default exception handler.

27

28 [*Note:* For a description of the exception handling model, see Partition I of the CLI  
29 Specification.]

30

31

32

33 [*Note:* If an application handles exceptions that occur during the execution of a block of  
34 application code, the code is required to be placed within a `try` statement. Application  
35 code within a `try` statement is a *try block*. Application code that handles Exceptions  
36 thrown by a `try` block is placed within a `catch` statement, and is called a *catch block*.  
37 Zero or more catch blocks are associated with a `try` block, and each catch block includes  
38 a type filter that determines the types of Exceptions it handles.

39

40 When an Exception occurs in a `try` block, the system searches the associated catch  
41 blocks in the order they appear in application code, until it locates a catch block that  
42 handles the Exception. A catch block handles an exception of type *T*, if the type filter of

1 the catch block specifies *T* or any type that *T* derives from. The system stops searching  
2 after it finds the first catch block that handles the Exception. For this reason, in  
3 application code, a catch block that handles a type must be specified before a catch  
4 block that handles its base types, as demonstrated in the example that follows this  
5 section. A catch block that handles `System.Exception` is specified last.

6  
7 If the catch blocks associated with the current try block do not handle the Exception,  
8 and the current try block is nested within other try blocks in the current call, the catch  
9 blocks associated with the next enclosing try block are searched. If no catch block for  
10 the Exception is found, the system searches previous nesting levels in the current call. If  
11 no catch block for the Exception is found in the current call, the Exception is passed up  
12 the call stack, and the previous stack frame is searched for a catch block that handles  
13 the Exception. The search of the call stack continues until the Exception is handled or  
14 there are no more frames in the call stack. If the top of the call stack is reached without  
15 finding a catch block that handles the Exception, the default exception handler handles it  
16 and the application terminates.

17  
18 ]

19  
20 `System.Exception` types support the following features:

- 21 • Human-readable text that describes the error. [*Note:* See  
22 `System.Exception.Message` property.]
- 23 • The state of the call stack when the Exception was thrown. [*Note:* See the  
24 `System.Exception.StackTrace` property.]
- 25 • When there is a causal relationship between two or more Exceptions, this information  
26 is maintained via the `System.Exception.InnerException` property.

27 The Base Class Library provides two types that inherit directly from `System.Exception`:

- 28 • `System.ApplicationException`
- 29 • `System.SystemException`

30 [*Note:* Most user-defined exceptions derive from `System.ApplicationException`. For more  
31 information, see `System.ApplicationException` and `System.SystemException`.]

32  
33

## 34 **Example**

35 The following example demonstrates a catch block that is defined to handle  
36 `System.ArithmeticException` errors. This catch block also catches  
37 `System.DivideByZeroException` errors because `System.DivideByZeroException`  
38 derives from `System.ArithmeticException`, and there is no catch block explicitly  
39 defined for `System.DivideByZeroException` errors.

40  
41 [C#]

```
1 using System;
2 class ExceptionTestClass {
3     public static void Main() {
4         int x = 0;
5         try {
6             int y = 100/x;
7         }
8         catch (ArithmeticException e) {
9             Console.WriteLine("ArithmeticException Handler: {0}", e.ToString());
10        }
11        catch (Exception e) {
12            Console.WriteLine("Generic Exception Handler: {0}", e.ToString());
13        }
14    }
15 }
```

17 The output is

```
18
19 ArithmeticException Handler: System.DivideByZeroException: Attempted to
20 divide by zero.
21     at ExceptionTestClass.Main()
22
23
```

# 1 Exception() Constructor

```
2 [ILAsm]  
3 public rtspecialname specialname instance void .ctor()  
4 [C#]  
5 public Exception()
```

## 6 Summary

7 Constructs and initializes a new instance of the `System.Exception` class.

## 8 Description

9 This constructor initializes the `System.Exception.Message` property of the new instance  
10 to a system-supplied message that describes the error and takes into account the  
11 current system culture. The `System.Exception.InnerException` property is initialized  
12 to null and the `System.Exception.StackTrace` property is initialized to  
13 `System.String.Empty`.

14

# 1 Exception(System.String) Constructor

```
2 [ILAsm]  
3 public rtspecialname specialname instance void .ctor(string message)  
4 [C#]  
5 public Exception(string message)
```

## 6 Summary

7 Constructs a new instance of the `System.Exception` class.

## 8 Parameters

Parameter	Description
<i>message</i>	A <code>System.String</code> that describes the error. The content of <i>message</i> is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

9

## 10 Description

11 This constructor initializes the `System.Exception.Message` property of the new instance  
12 using *message*. If *message* is null, the `System.Exception.Message` property is  
13 initialized to the system-supplied message provided by the constructor that takes no  
14 arguments. The `System.Exception.InnerException` property is initialized to null and  
15 the `System.Exception.StackTrace` property is initialized to `System.String.Empty`.

16

# 1 Exception(System.String, System.Exception) 2 Constructor

```
3 [ILAsm]  
4 public rtspecialname specialname instance void .ctor(string message, class  
5 System.Exception innerException)  
  
6 [C#]  
7 public Exception(string message, Exception innerException)
```

## 8 Summary

9 Constructs a new instance of the System.Exception class.

## 10 Parameters

Parameter	Description
<i>message</i>	A System.String that describes the error. The content of <i>message</i> is intended to be understood by humans. The caller of this constructor is required to ensure that this string has been localized for the current system culture.
<i>innerException</i>	An instance of System.Exception that is the cause of the current exception. If <i>innerException</i> is non-null, then the current exception was raised in a catch block handling <i>innerException</i> .

## 11 12 Description

13 This constructor initializes the System.Exception.Message property of the new instance  
14 using *message*, and the System.Exception.InnerException property using  
15 *innerException*. If *message* is null, the System.Exception.Message property is  
16 initialized to the system-supplied message provided by the constructor that takes no  
17 arguments.

18  
19 The System.Exception.StackTrace property is initialized to System.String.Empty.

20

# 1 Exception.GetBaseException() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual class System.Exception GetBaseException()  
4 [C#]  
5 public virtual Exception GetBaseException()
```

## 6 Summary

7 Returns the `System.Exception` that is the root cause of one or more subsequent  
8 Exceptions.

## 9 Return Value

10 Returns the first Exception thrown in a chain of Exceptions. If the  
11 `System.Exception.InnerException` property of the current Exception is null, returns  
12 the current Exception.

## 13 Description

14 [Note: A chain of Exceptions consists of a set of Exceptions such that each Exception in  
15 the chain was thrown as a direct result of the Exception referenced in its  
16 `System.Exception.InnerException` property. For a given chain, there can be exactly  
17 one Exception that is the root cause of all other Exceptions in the chain. This Exception  
18 is called the *baseexception* and its `System.Exception.InnerException` property always  
19 contains a null reference.]  
20  
21

## 22 Behaviors

23 For all Exceptions in a chain of Exceptions, the `System.Exception.GetBaseException`  
24 method is required to return the same object (the *base exception* ).  
25

## 26 How and When to Override

27 The `System.Exception.GetBaseException` method is overridden in classes that require  
28 control over the exception content or format.  
29

## 30 Usage

31 Use the `System.Exception.GetBaseException` method when you want to find the root  
32 cause of an Exception but do not need information about Exceptions that might have  
33 occurred between the current Exception and the first Exception.

1

## 2 **Example**

3     The following example shows an implementation of the  
4     System.Exception.GetBaseException method.

5

6     [C#]

```
7 public virtual Exception GetBaseException() {  
8     Exception inner = InnerException;  
9     Exception back = this;  
10    while (inner != null) {  
11        back = inner;  
12        inner = inner.InnerException;  
13    }  
14    return back;  
15 }  
16  
17
```

# 1 Exception.ToString() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual string ToString()  
4 [C#]  
5 public override string ToString()
```

## 6 Summary

7 Creates and returns a `System.String` representation of the current `Exception`.

## 8 Return Value

9 A `System.String` representation of the current `Exception`.

## 10 Behaviors

11 `System.Exception.ToString` returns a representation of the current `Exception` that is  
12 intended to be understood by humans. Where the `Exception` contains culture-sensitive  
13 data, the string representation returned by `System.Exception.ToString` is required to  
14 take into account the current system culture. [*Note:* Although there are no exact  
15 requirements for the format of the returned string, it should as much as possible reflect  
16 the value of the object as perceived by the user.]

17  
18  
19  
20 [*Note:* This method overrides `System.Object.ToString`.]  
21  
22

## 23 Default

24 The `System.Exception.ToString` implementation obtains the fully qualified name of the  
25 current `Exception`, the message, the result of calling `System.Exception.ToString` on  
26 the inner exception, and the result of calling `System.Environment.StackTrace`. If any  
27 of these members is null or equal to `System.String.Empty`, its value is not included in  
28 the returned string.

## 29 How and When to Override

30 It is recommended, but not required, that `System.Exception.ToString` be overridden  
31 to return information about members declared in the derived class. For example, the  
32 `System.ArgumentException` class overrides `System.Exception.ToString` so that it  
33 returns the value of the `System.ArgumentException.ParamName` property, if that value  
34 is not null.

## 35 Usage

1 Use the `System.Exception.ToString` method to obtain a string representation of an  
2 Exception.

### 3 **Example**

4 The following example causes an Exception and displays the result of calling  
5 `System.Exception.ToString` on that Exception.

```
6 [C#]
7
8 using System;
9 public class MyClass {}
10 public class ArgExceptionExample {
11     public static void Main() {
12         MyClass my = new MyClass();
13         string s = "sometext";
14         try {
15             int i = s.CompareTo(my);
16         }
17         catch (Exception e) {
18             Console.WriteLine("Error: {0}", e.ToString());
19         }
20     }
21 }
```

22 The output is

```
24
25 Error: System.ArgumentException: Object must be of type String.
26     at System.String.CompareTo(Object value)
27     at ArgExceptionExample.Main()
28
29
```

# 1 Exception.InnerException Property

```
2 [ILAsm]
3 .property class System.Exception InnerException { public hidebyref
4 specialname instance class System.Exception get_InnerException() }
5 [C#]
6 public Exception InnerException { get; }
```

## 7 Summary

8 Gets the `System.Exception` instance that caused the current `Exception`.

## 9 Property Value

10 An instance of `System.Exception` that describes the error that caused the current  
11 `Exception`.

## 12 Description

13 This property is read-only.

14  
15 [*Note:* When an `Exception X` is thrown as a direct result of a previous exception `Y`, the  
16 `System.Exception.InnerException` property of `X` should contain a reference to `Y`.]  
17

18  
19  
20 The `System.Exception.InnerException` property returns the same value as was  
21 passed into the constructor, or `null` if the inner exception value was not supplied to the  
22 constructor.  
23

24 [*Note:* Using the `System.Exception.InnerException` property, you can obtain the set  
25 of `Exceptions` that led to the current `Exception`. `System.Exception.GetBaseException`  
26 includes an example that demonstrates this procedure.]  
27  
28

## 29 Example

30 The following example demonstrates throwing and catching an `Exception` that references  
31 an inner `Exception`.  
32

```
33 [C#]
34 using System;
35 public class MyAppException:ApplicationException {
36     public MyAppException (String message): base (message) {}
37     public MyAppException (String message, Exception inner): base(message,inner)
38     {}
39 }
40 public class ExceptExample {
```

```

1  public void ThrowInner () {
2  throw new MyAppException("ExceptExample inner exception");
3  }
4  public void CatchInner() {
5  try {
6  this.ThrowInner();
7  }
8  catch (Exception e) {
9  throw new MyAppException("Error caused by trying ThrowInner.",e);
10 }
11 }
12 }
13 public class Test {
14 public static void Main() {
15 ExceptExample testInstance = new ExceptExample();
16 try {
17 testInstance.CatchInner();
18 }
19 catch(Exception e) {
20 Console.WriteLine ("In Main catch block. Caught: {0}", e.Message);
21 Console.WriteLine ("Inner Exception is {0}",e.InnerException);
22 }
23 }
24 }

```

25  
26 The output is

```

27
28 In Main catch block. Caught: Error caused by trying ThrowInner.
29 Inner Exception is MyAppException: ExceptExample inner exception
30 at ExceptExample.ThrowInner()
31 at ExceptExample.CatchInner()
32
33

```

# 1 Exception.Message Property

```
2 [ILAsm]  
3 .property string Message { public hidebysig virtual specialname string  
4 get_Message() }  
5 [C#]  
6 public virtual string Message { get; }
```

## 7 Summary

8 Gets a `System.String` containing a message that describes the current Exception.

## 9 Property Value

10 A `System.String` that contains a detailed description of the error, or  
11 `System.String.Empty`. This value is intended to be understood by humans.

## 12 Description

13 [*Note:* The text of `System.Exception.Message` should completely describe the error and  
14 should, when possible, explain how to correct it.

15  
16 The value of the `System.Exception.Message` property is included in the information  
17 returned by `System.Exception.ToString`.

18 ]

19  
20  
21 This property is read-only.

## 22 Behaviors

23 The `System.Exception.Message` property is set only when creating an Exception  
24 instance.

25  
26 If no message was supplied to the constructor for the current instance, the system  
27 supplies a default message that is formatted using the current system culture.

## 28 How and When to Override

29 The `System.Exception.Message` property is overridden in classes that require control  
30 over message content or format.

## 31 Usage

32 Application code typically accesses this property when there is a need to display  
33 information about an exception that has been caught.

34

# 1 Exception.StackTrace Property

```
2 [ILAsm]  
3 .property string StackTrace { public hidebysig virtual specialname string  
4 get_StackTrace() }  
  
5 [C#]  
6 public virtual string StackTrace { get; }
```

## 7 Summary

8 Gets a `System.String` representation of the frames on the call stack at the time the  
9 current Exception was thrown.

## 10 Property Value

11 A `System.String` that describes the contents of the call stack.

## 12 Description

13 [*Note:* `System.Exception.StackTrace` might not report as many method calls as  
14 expected, due to code transformations, such as inlining, that occur during optimization.]  
15  
16  
17

18 This property is read-only.

## 19 Behaviors

20 The format of the information returned by this property is required to be identical to the  
21 format of the information returned by `System.Environment.StackTrace`.

## 22 How and When to Override

23 The `System.Exception.StackTrace` property is overridden in classes that require  
24 control over the stack trace content or format.

## 25 Usage

26 Use the `System.Exception.StackTrace` property to obtain a string representation of  
27 the contents of the call stack at the time the exception was thrown.  
28