**NullReferenceException**

What is a possible situation that leads to the exception?

This exception occurs when caller tries to access a member while object is null. A NullReferenceException exception is caused by developer error and thrown in this scenario:

When programmer forgets to instantiate a reference this exception occurs. Like

using System;

using System.Collections.Generic;

public class Example

{

public static void Main(string[] args)

{

int value = Int32.Parse(args[0]);

List<String> names;

if (value > 0)

names = new List<String>();

names.Add("Major Major Major");

}

}

Who is in charge of throwing the exception: you as a programmer or the runtime system?

In this particular exception **runtime System** **is in charge** of throwing this exception. So

Compilation displays a warning like:

Variable names is used before it has been assigned a value. A null reference exception could result at runtime.

names.Add("Major Major Major")

The example displays output like the following:

Unhandled Exception: System.NullReferenceException: Object reference not set to an instance of an object at Example.Main()

In theory, should you throw exceptions of this type? Explain your answer.

No, we should not through exceptions of this type but we should avoid the situation of this type. These exceptions (reserved) thrown by the system and not by the programmers. It is one of the exceptions reserved by the system.

If you need to throw the exception, what details would you provide as a message to the user (caller)? What parameters would you include in the message?

Firstly, I should not throw this exception because it is one of the system-reserved exceptions but still if I need to throw this exception, I would provide a message that points directly to the error occurring line and information about the cause of the error.

Can the exception be generally caught (and therefore handled)?

Yes, exception can be easily caught and handled to avoid errors that could occur later.

**For Example:**

using System;

using System.Collections.Generic;

public class Example

{

public static void Main(string[] args)

{

int value = Int32.Parse(args[0]);

List<String> names = new List<String>();

try

{

if (value > 0)

names = new List<String>();

names.Add("Major Major Major");

}catch (NullReferenceException ex)

{

Console.WriteLine (ex.Message);

}

}

}

If the exception occurs, should you generally catch this exception type or is it better to pass such exception to the user (caller)? It is not enough to say yes or no; you must provide an argument to support your answer.

No I will not catch this exception but only thing I will do is avoiding the situation that caused this exception. Because it is not better to crash, the normal flow of the program at a huge cost of exception and then catch it. As we can check the reference not to be null before accessing its class members and secondly we can first instantiate it and then access its data members.

If I am intentionally throwing this exception then I will surely catch it and secondly if this is caused by system then it should be passed to the Caller.

Like:

using System;

using System.Collections.Generic;

public class Example

{

public static void Main()

{

List<String> names = new List<String>();

names.Add("Major Major Major");

}

}

Is the exception a case when you want to avoid this exception to occur in your application in general? If so, what would be your actions as a programmer to avoid it?

Yes, it is a different case when I try to avoid it.

I will check this like

* Initializing the object and then accessing the class members

using System;

public class Example

{

public static void Main()

{

int[] values = new int[10];

for (int i = 0; i <= 9; i++)

values[i] = i \* 2;

foreach (var value in values)

Console.WriteLine(value);

}

}

**IndexOutOfRangeException**

What is a possible situation that leads to the exception?

The exception that is thrown when an attempt is made to access an element of an array or collection with an index that is outside its boundary.

Forgetting that the upper bound of a collection or a zero-based array is one less than its number of members or elements, as the following example illustrates. Like

using System;

using System.Collections.Generic;

public class Example

{

public static void Main()

{

List<Char> characters = new List<Char>();

characters.InsertRange(0, new Char[] { 'a', 'b', 'c', 'd', 'e', 'f' } );

for (int i = 0; i <= characters.Count; i++)

Console.Write("'{0}' ", characters[ctr]);

}

}

The example displays the following output and an exception:

'a' 'b' 'c' 'd' 'e' 'f'

Unhandled Exception:

System.ArgumentOutOfRangeException:

Index was out of range. Must be non-negative and less than the size of the collection.

Parameter name: index

at Example.Main()

Who is in charge of throwing the exception: you as a programmer or the runtime system?

In this particular exception, **runtime-System** **is in charge** of throwing this exception. So

Compilation displays a message like:

Unhandled Exception:

System.ArgumentOutOfRangeException:

Index was out of range. Must be non-negative and less than the size of the collection.

Parameter name: index

at Example.Main()

In theory, should you throw exceptions of this type? Explain your answer.

No, we should not through exceptions of this type but we should avoid the cause of this exception. These exceptions (reserved) thrown by the system and not by the programmers but programmer causes it. It is one of the exceptions reserved by the system and system throws this exception.

If you need to throw the exception, what details would you provide as a message to the user (caller)? What parameters would you include in the message?

Firstly, I should not throw this exception because it is one of the system-reserved exceptions but still if I need to throw this exception, I would provide a message that points directly to the error occurring line and information about the cause of the error.

Can the exception be generally caught (and therefore handled)?

No, we cannot catch this and handle. We should **validate** the variable that is used for accessing array values to avoid this situation.

**For Example:**

using System;

using System.Collections.Generic;

public class Example

{

public static void Main()

{

List<Char> characters = new List<Char>();

characters.InsertRange(0, new Char[] { 'a', 'b', 'c', 'd', 'e', 'f' } );

for (int i = 0; i < characters.Count; i++)

Console.Write("'{0}' ", characters[i]);

}

}

The example displays the following output:

'a' 'b' 'c' 'd' 'e' 'f'

If the exception occurs, should you generally catch this exception type or is it better to pass such exception to the user (caller)? It is not enough to say yes or no; you must provide an argument to support your answer.

No I will not catch this exception but rather than catching, I will add a new check to insure I am accessing the array values within the array range. As we can check the array indexes to remain in their bounds, so we will avoid this exception rather than catching it.

Is the exception a case when you want to avoid this exception to occur in your application in general? If so, what would be your actions as a programmer to avoid it?

Yes, it is a different case when I try to avoid it.

I will check this like

* Better validating the traversal of array indexes

Like:

Line that caused exception

for (int i = 0; i <= characters.Count; i++)

Line that avoided exception

for (int i = 0; i < characters.Count; i++)

**StackOverflowException**

What is a possible situation that leads to the exception?

The exception that is thrown when the execution stack (that stores chain calls) overflows because it contains too many-nested method calls. This class cannot be inherited.

When your code is very deep or contains boundless recursion. So make sure your code doesn't have any infinite loop or infinite recursion.

Like

using System;

public class Example

{

private const int MAX\_RECURSIVE\_CALLS = 1000;

static int ctr = 0;

public static void Main()

{

Example ex = new Example();

ex.Execute();

Console.WriteLine("\nThe call counter: {0}", ctr);

}

private void Execute()

{

ctr++;

if (ctr % 50 == 0)

Console.WriteLine("Call number {0} to the Execute method", ctr);

Execute();

ctr--;

}

}

The example displays the following output:

Call number 50 to the Execute method

Call number 100 to the Execute method

Call number 150 to the Execute method

Call number 200 to the Execute method

Call number 250 to the Execute method

Call number 300 to the Execute method

Call number 350 to the Execute method

Call number 400 to the Execute method

Call number 450 to the Execute method

Call number 500 to the Execute method

Call number 550 to the Execute method

Call number 600 to the Execute method

Call number 650 to the Execute method

Call number 700 to the Execute method

Call number 750 to the Execute method

Call number 800 to the Execute method

Call number 850 to the Execute method

Call number 900 to the Execute method

Call number 950 to the Execute method

Call number 1000 to the Execute method

After the Stack Limit

Unhandled Exception:

System.StackOverFlowException

Who is in charge of throwing the exception: you as a programmer or the runtime system?

In this particular exception **runtime System** **is in charge** of throwing this exception. So

Runtime displays a message like:

System.StackOverFlowException

names.Add("Major Major Major")

The example displays output like the following:

Unhandled Exception: System. StackOverFlowException:

In theory, should you throw exceptions of this type? Explain your answer.

No, we should not through exceptions of this type but we should add a base case to avoid this exception in recursive methods and in infinite loops. When user’s chain calls are too much that it consumes the stack limit (that is used by the system to handle method calls to keep track) then StackOverFlowException occurs It is one of the exceptions reserved by the system.

If you need to throw the exception, what details would you provide as a message to the user (caller)? What parameters would you include in the message?

I should not throw this exception because I am not allowed to throw this type of exception it is system-reserved exception.

Can the exception be generally caught (and therefore handled)?

No, this exception cannot be caught but can be avoided by providing the base case in the program. Applying the “HandleProcessCorruptedStateExceptionsAttribute” attribute to a method that throws a StackOverflowException has no effect. You still cannot handle the exception from user’s code.

If the exception occurs, should you generally catch this exception type or is it better to pass such exception to the user (caller)? It is not enough to say yes or no; you must provide an argument to support your answer.

No I cannot catch this exception starting with the .NET Framework 2.0, user cannot catch a StackOverflowException object with a try/catch block, and the corresponding process is terminated by default. Consequently, we should have to write our code to detect and prevent a stack overflow exception. For example, if our app depends on recursion, **we should provide a base case to end the recursive calls.**

Is the exception a case when you want to avoid this exception to occur in your application in general? If so, what would be your actions as a programmer to avoid it?

Yes, it is better to avoid this exception rather than catching it. Referring to .NET Framework 2.0, user cannot catch a StackOverflowException object with a try/catch block and the corresponding process is terminated by default. If our app hosts the common language runtime (CLR), it can specify that the CLR should unload the application domain where the stack overflow exception occurs and let the corresponding process continue. **In this way, we can avoid it.**

**OutOfMemoryException**

What is a possible situation that leads to the exception?

The exception that is thrown when there is not enough memory to continue the execution of a program.

* You are attempting to expand a StringBuilder object beyond the length defined by its StringBuilder.MaxCapacity property.
* The common language runtime cannot allocate enough contiguous memory to perform an operation.

**For Example:**

using System;

using System.Text;

public class Program

{

public static void Main()

{

StringBuilder stringBuilder = new StringBuilder(17, 17);

stringBuilder.Append("Welcome to the ");

try

{

stringBuilder.Insert(0, "world of C# programming", 1);

Console.WriteLine(stringBuilder.ToString());

Console.ReadLine();

}

catch (OutOfMemoryException exception)

{

Console.WriteLine(exception.Message);

Console.ReadLine();

}

}

}

Who is in charge of throwing the exception: you as a programmer or the runtime system?

In this particular exception **runtime System** **is in charge** of throwing this exception.

In theory, should you throw exceptions of this type? Explain your answer.

No, we should not through exceptions of this type. However, if I need to handle the memory I may have need to throw this exception in **defensive cases**. As in 32-bit system, 2MB is allocated to a program while in 64bit 4MB is allocated. An OutOfMemoryException does not mean that the memory is completely depleted it just means that a memory allocation failed. If you tried to allocate a large memory area at once, there may still be plenty of free memory left. So memory allocation is handled by the system and hence if any exception related to memory occurs then system throws that exception.

If you need to throw the exception, what details would you provide as a message to the user (caller)? What parameters would you include in the message?

If I need to throw this exception, I would provide “Out of memory message to the Caller” so that he may better understand about how to handle this case.

Can the exception be generally caught (and therefore handled)?

Yes, exception can be easily caught and handled to avoid errors that could occur later due to insufficient memory to execute the program.

**For Example:**

using System;

using System.Text;

public class Example

{

public static void Main()

{

StringBuilder sb = new StringBuilder(15, 15);

sb.Append("Substring #1 ");

try {

sb.Insert(0, "Substring #2 ", 1);

}

catch (OutOfMemoryException e) {

Console.WriteLine("Out of Memory: {0}", e.Message);

}

}

}

If the exception occurs, should you generally catch this exception type or is it better to pass such exception to the user (caller)? It is not enough to say yes or no; you must provide an argument to support your answer.

Yes, I should catch the exception and continue the normal flow of the program.

If programmer throws this exception then I will surely catch it and secondly if this is caused by runtime system due to expanding string buffer then it should be passed to the Caller.

Like:

Is the exception a case when you want to avoid this exception to occur in your application in general? If so, what would be your actions as a programmer to avoid it?

Yes, to avoid this exception while working with StringBuilder, we can call the constructor of the StringBuilder. **StringBuilder (Int32, Int32) and can set the MaxCapacity property to a value that will be large enough to serve the accommodation required when we expand the corresponding StringBuilder object.**

**DivideByZeroException**

What is a possible situation that leads to the exception?

The exception that is thrown when there is an attempt to divide an integral or decimal value by zero.

**For Example:**

using System;

namespace ErrorHandlingApplication {

class DivNumbers {

int result;

DivNumbers() {

result = 0;

}

public void division(int num1, int num2) {

result = num1 / num2;

}

static void Main(string[] args) {

DivNumbers d = new DivNumbers();

d.division(25, 0);

Console.ReadKey();

}

}

}

Who is in charge of throwing the exception: you as a programmer or the runtime system?

In this particular exception **runtime System** **is in charge** of throwing this exception. This exception arises when user have not validated all of the user inputs and user enters a value that leads to the exception.

In theory, should you throw exceptions of this type? Explain your answer.

Yes, we should through exceptions of this type.

**For Example:**

If we are developing a calculator application then we need to handle several validation checks for proper working of the application. We can use this exception here and can later handle it. When user tries to divide by zero then this exception is helpful.

If you need to throw the exception, what details would you provide as a message to the user (caller)? What parameters would you include in the message?

If I need to throw this exception, I would provide “Cannot Divide by Zero” as a message so that he may better understand the error. In addition, I would provide ex.Message to the user.

Can the exception be generally caught (and therefore handled)?

Yes, it can be handled to avoid error. Further we can provide message to the user for further proceeding to solve the error. Following code handles the divide by zero exception.

**For Example:**

using System;

namespace ErrorHandlingApplication {

class DivNumbers {

int result;

DivNumbers() {

result = 0;

}

public void division(int num1, int num2) {

try {

result = num1 / num2;

} catch (DivideByZeroException e) {

Console.WriteLine("Exception caught: {0}", e);

} finally {

Console.WriteLine("Result: {0}", result);

}

}

static void Main(string[] args) {

DivNumbers d = new DivNumbers();

d.division(25, 0);

Console.ReadKey();

}

}

}

If the exception occurs, should you generally catch this exception type or is it better to pass such exception to the user (caller)? It is not enough to say yes or no; you must provide an argument to support your answer.

Yes, I should catch this exception and I will provide clear message to the user instead of breaking the program normal flow.

**For Example:** Considering the example of calculator application when this exception occurs if we directly pass this exception to the user. User uses the application to perform the needed operation if he encounters this exception then he cannot perform the needed operation. Hence it is not better to pass this to user instead I should catch this exception and provide a clear message to the user.

Is the exception a case when you want to avoid this exception to occur in your application in general? If so, what would be your actions as a programmer to avoid it?

Yes, I need to avoid this exception and in order to prevent the exception, ensure that the denominator in a division operation with integer or Decimal values is non-zero. I would have to validate the user input before passing it to the division method.

**For Example:**

public void division(int num1, int num2) {

try {

if (num2 > 0)

{

result = num1 / num2;

}

} catch (DivideByZeroException e) {

Console.WriteLine("Exception caught: {0}", e);

} finally {

Console.WriteLine("Result: {0}", result);

}

}

**ArgumentNullException**

What is a possible situation that leads to the exception?

The exception that is thrown when a **null reference** is passed to a method that does not accept it as a valid argument.

**For Example:**

namespace Example {

public class ExExample{

public void MyMethod(string text) {

//Throws exception if text is equal to null

if (text.ToUpper() == “Hello World”)

{

Console.WriteLine(text);

}

}

Public static void main(string[] args){

MyMethod(null);

}

}

}

Who is in charge of throwing the exception: you as a programmer or the runtime system?

In this particular exception, **runtime-System** **is in charge** of throwing this exception. This exception arises when user have not validated object references before calling any data members if this reference is null and it is came as an argument to the method then it arises nullreferenceexception.

In theory, should you throw exceptions of this type? Explain your answer.

No, we should not through exceptions of this type.

**For Example:**

The ArgumentNullException is thrown when a null value is passed to a method that does not accept null values as valid input. They are provided so that application code can differentiate between exceptions caused by null arguments and exceptions caused by arguments that are not null. Therefore, we should not through exceptions of this type. However, we can use it in defensive cases where programmer checks for invalid values.

If you need to throw the exception, what details would you provide as a message to the user (caller)? What parameters would you include in the message?

I would provide “Argument object is null please instantiate” as a message. In addition, I would provide ex.Message as another argument.

Can the exception be generally caught (and therefore handled)?

Yes, it can be handled to avoid breakage of the program. Further, we can provide message to the user without losing the flow of the program. Following code handles the null reference exception.

**For Example:**

namespace Example {

public class ExExample{

public void MyMethod(string text) {

//Throws exception if text is equal to null

try

{

if (text.ToUpper() == “Hello World”)

{

Console.WriteLine(text);

}

}catch (NullReferenceException ex)

{

Console.WriteLine(“Argument object is null please instantiate”, ex.Message);

}

}

Public static void main(string[] args){

MyMethod(null);

}

}

}

If the exception occurs, should you generally catch this exception type or is it better to pass such exception to the user (caller)? It is not enough to say yes or no; you must provide an argument to support your answer.

Yes, I should catch this exception and I will provide clear message to the user rather than disturbing the normal flow of the user. So that he may differentiate between exception caused by null or not null value.

**For Example:** ArgumentNullException behaves identically to ArgumentException. It is provided so that the application code can differentiate between exceptions caused by null arguments and exceptions caused by arguments that are not null. Therefore, we should pass this exception to the user.

Is the exception a case when you want to avoid this exception to occur in your application in general? If so, what would be your actions as a programmer to avoid it?

Yes, I need to avoid this exception and in order to prevent the exception, ensure that users provide valid values. I would have to validate the user input before accessing its methods.

**Actions to Avoid:**

* Initialize variables with valid values.
* If a variable can be null, then check for null and handle it appropriately
* Use the “?” operator on methods when possible. stringvar?.ToUpper();
* Use tools like Resharper to help point out potential null reference exceptions

**ArgumentOutOfRangeException**

What is a possible situation that leads to the exception?

The exception that is thrown when the value of an argument is outside the allowable range of values as defined by the invoked method. An ArgumentOutOfRangeException exception is thrown when a method is invoked and at least one of the arguments passed to the method is not null and contains an invalid value that is not a member of the set of values expected for the argument. The ParamName property identifies the invalid argument, and the ActualValue property, if a value is present, identifies the invalid value.

**For Example:**

using System;

class Program

{

static void Main(string[] args)

{

try

{

Guest guest1 = new Guest("Ben", "Miller", 17);

Console.WriteLine(guest1.GuestInfo());

}

catch (ArgumentOutOfRangeException outOfRange)

{

Console.WriteLine("Error: {0}", outOfRange.Message);

}

}

}

class Guest

{

private string FirstName;

private string LastName;

private int Age;

public Guest(string fName, string lName, int age)

{

FirstName = fName;

LastName = lName;

if (age < 21)

throw new ArgumentOutOfRangeException("age","All guests must be 21-years-old or older.");

else

Age = age;

}

public string GuestInfo()

{

string gInfo = FirstName + " " + LastName + ", " + Age.ToString();

return(gInfo);

}

}

Who is in charge of throwing the exception: you as a programmer or the runtime system?

Typically, an ArgumentOutOfRangeException results from **programmer error**. Instead of handling the exception in a try/catch block, we should eliminate the cause of the exception or, if the argument is returned by a method call or input by the user before being passed to the method that throws the exception, you should validate arguments before passing them to the method.

In theory, should you throw exceptions of this type? Explain your answer.

Yes, I should throw my own, for a couple of reasons:

I can explicitly set the appropriate parameter name in the constructor. This way, the exception has the appropriate parameter information for the Argument that is out of range. The internal list's exception will have an invalid stack trace as far as your user is concerned. So I should throw my own “ArgumentOutOfRangeException” for our own parameters.

If you need to throw the exception, what details would you provide as a message to the user (caller)? What parameters would you include in the message?

I would provide “Please add only allowed parameters” as a message. In addition, I would provide ex.Message as another argument.

Can the exception be generally caught (and therefore handled)?

Yes, it can be handled for safe exit of program. Instead of handling the exception in a try/catch block, we should eliminate the cause of the exception or, if the argument is returned by a method call or input by the user before being passed to the method that throws the exception, you should validate arguments before passing them to the method. Further, we can provide message to the user without losing the flow of the program.

**For Example:**

using System;

using System.Collections.Generic;

public class Example

{

public static void Main()

{

var list = new List<string>();

Console.WriteLine("Number of items: {0}", list.Count);

try {

Console.WriteLine("The first item: '{0}'", list[0]);

}

catch (ArgumentOutOfRangeException e) {

Console.WriteLine(e.Message);

}

}

}

The example displays the following output:

Number of items: 0

Index was out of range. Must be non-negative and less than the size of the collection.

Parameter name: index

If the exception occurs, should you generally catch this exception type or is it better to pass such exception to the user (caller)? It is not enough to say yes or no; you must provide an argument to support your answer.

Instead of handling the exception in a try/catch block, we should eliminate the cause of the exception or, if the argument is returned by a method call or input by the user before being passed to the method that throws the exception, we should validate arguments before passing them to the method. Since this exception tells user about wrong arguments of the method that were designed to accept specific inputs So we need to pass this exception to the user so that he may understand which arguments are wrong.

Is the exception a case when you want to avoid this exception to occur in your application in general? If so, what would be your actions as a programmer to avoid it?

Yes, I need to avoid this exception and in order to prevent the exception, check that the search is successful by making sure that the returned index is greater than or equal to zero before attempting to retrieve the item from the collection, as the following code fragment does.

**For Example:**

Get the index of the element whose value is "Z".

int index = list.FindIndex((new StringSearcher("Z")).FindEquals);

if (index >= 0)

Console.WriteLine("'Z' is found at index {0}", list[index]);

**FormatException**

What is a possible situation that leads to the exception?

The exception that is thrown when the format of an argument is invalid, or when a composite format string is not well formed. In a call to a method that converts a string to some other data type, the string does not conform to the required pattern. This typically occurs when calling some methods of the Convert class and the Parse and ParseExact methods of some types.

**For Example:**

using System;

public class Example

{

public static void Main()

{

decimal price = 169.32m;

Console.WriteLine("The cost is {0:Q2}.", price);

}

}

The example displays the following output:

Unhandled Exception: System.FormatException: Format specifier was invalid.

at System.Number.FormatDecimal(Decimal value, String format, NumberFormatInfo info)

at System.Decimal.ToString(String format, IFormatProvider provider)

at System.Text.StringBuilder.AppendFormat(IFormatProvider provider, String format, Object[] args)

at System.IO.TextWriter.WriteLine(String format, Object arg0)

at System.IO.TextWriter.SyncTextWriter.WriteLine(String format, Object arg0)

at Example.Main()

Who is in charge of throwing the exception: you as a programmer or the runtime system?

Typically, an FormatException results from **programmer error**. In a call to a method that converts a string to some other data type, the string does not conform to the required pattern. This typically occurs when calling some methods of the Convert class and the Parse and ParseExact methods of some types.

In theory, should you throw exceptions of this type? Explain your answer.

No, I should not throw exceptions of this type. As I have built in methods like Parse, TryParse and others that are used for formatting and all of them through FormatException. Therefore, I do not have need to explicitly throwing the same exception rather I can use above described methods to perform our formatting operations.

If you need to throw the exception, what details would you provide as a message to the user (caller)? What parameters would you include in the message?

I would provide “Add formatting” as a message. In addition, I would provide ex.Message as another argument.

Can the exception be generally caught (and therefore handled)?

Yes, I can catch and then handle. This exception results from a coding error. To correct the error, either remove the format string or substitute a valid one. The following example corrects the error by replacing the invalid format string with the "C" (currency) format string.

**For Example:**

using System;

public class Example

{

public static void Main()

{

decimal price = 169.32m;

Console.WriteLine("The cost is {0:C2}.", price);

}

}

The example displays the following output:

The cost is $169.32.

If the exception occurs, should you generally catch this exception type or is it better to pass such exception to the user (caller)? It is not enough to say yes or no; you must provide an argument to support your answer.

No, it is better to pass it to the caller. In this case, the FormatException exception is a result of programmer error. It should be corrected rather than handled in a try/catch block by making sure that each item in the object list corresponds to the index of a format item. In order to correct this example, change the index of the second format item to refer to the dat variable, and decrement the index of each subsequent format item by one. Therefore, if programmer does not correct the item from the object list to make sure it is format item then I should pass this exception to the user.

Is the exception a case when you want to avoid this exception to occur in your application in general? If so, what would be your actions as a programmer to avoid it?

In most cases, particularly if the string that you are converting is input by a user or read from a file, you should use a try/catch block and handle the FormatException exception if the conversion is unsuccessful. You can also replace the call to the conversion method with a call to a TryParse or TryParseExact method, if one exists. However, a FormatException exception that is thrown when you are trying to parse a predefined or hard-coded string indicates a program error. In this case, you should correct the error rather than handle the exception.

**For Example:**

using System;

public class Example

{

public enum Temperature Scale

{ Celsius, Fahrenheit, Kelvin }

public static void Main()

{

String info = GetCurrentTemperature();

Console.WriteLine(info);

}

private static String GetCurrentTemperature()

{

DateTime dat = DateTime.Now;

Decimal temp = 20.6m;

TemperatureScale scale = TemperatureScale.Celsius;

String result;

result = String.Format("At {0:t} on {0:D}, the temperature is {1:F1} {2:G}",

dat, temp, scale);

return result;

}

}

The example displays output like the following:

At 10:40 AM on Wednesday, June 04, 2014, the temperature is 20.6 Celsius

**SystemException**

What is a possible situation that leads to the exception?

This class is provided as a means to differentiate between system exceptions and application exceptions. It is the base class of such exceptions as ArgumentException, FormatException, and InvalidOperationException.

Since SystemException, serves as the base class of a variety of exception types, our code should not throw a SystemException exception, nor should it attempt to handle a SystemException exception unless I intend to re-throw the original exception.

Who is in charge of throwing the exception: you as a programmer or the runtime system?

This is base class of all exception that differentiate between system and application exception. Neither it is thrown nor it catches but it has several child classes that throw and capture exceptions. SYSTEM is in charge of its child exceptions.

In theory, should you throw exceptions of this type? Explain your answer.

No, I should not throw exceptions of this type because it serves as the base class of a variety of exception types, our code should not throw a SystemException exception, nor should it attempt to handle a SystemException exception unless you intend to re-throw the original exception.

If you need to throw the exception, what details would you provide as a message to the user (caller)? What parameters would you include in the message?

The content of the message parameter should be understandable to the user. The caller of this constructor is required to ensure that this string has been localized for the current system culture.

Can the exception be generally caught (and therefore handled)?

Yes, I can catch and then handle. This exception is the parent of all exceptions so I can use it to catch its child exceptions.

If the exception occurs, should you generally catch this exception type or is it better to pass such exception to the user (caller)? It is not enough to say yes or no; you must provide an argument to support your answer.

It is better to avoid the situation that creates this scenario. It is bad practice to catch System.Exception or better yet, it is bad practice to handle System.Exception anywhere but the top level of your application. What you should do is:

Catch System.Exception

Test the exception for the types you plan to handle identically

Re-throw if it is not one of those specific exceptions.

Is the exception a case when you want to avoid this exception to occur in your application in general? If so, what would be your actions as a programmer to avoid it?

As I discussed earlier SystemException is the base class of other child classes so in order to avoid first I need to check which specific type it contains in it so can provide possible actions to avoid this exception.