An exception is a problem that arises during the execution of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: **try**, **catch**, **finally**, and **throw**.

- **try**: A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.

- **catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

- **finally**: The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.

- **throw**: A program throws an exception when a problem shows up. This is done using a throw keyword.

**Syntax**

Assuming a block raises an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```csharp
try
{
    // statements causing exception
}
catch( ExceptionName e1 )
{
    // error handling code
}
catch( ExceptionName e2 )
{
    // error handling code
}
catch( ExceptionName eN )
{
    // error handling code
} finally
{
    // statements to be executed
}
```

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

**Exception Classes in C#**

C# exceptions are represented by classes. The exception classes in C# are mainly directly or indirectly derived from the **System.Exception** class. Some of the exception classes derived from the System.Exception class are the **System.ApplicationException** and **System.SystemException** classes.

The **System.ApplicationException** class supports exceptions generated by application
programs. Hence the exceptions defined by the programmers should derive from this class.

The **System.SystemException** class is the base class for all predefined system exception.

The following table provides some of the predefined exception classes derived from the System.SystemException class:

<table>
<thead>
<tr>
<th>Exception Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.IO.IOException</td>
<td>Handles I/O errors.</td>
</tr>
<tr>
<td>System.IndexOutOfRangeException</td>
<td>Handles errors generated when a method refers to an array index out of range.</td>
</tr>
<tr>
<td>System.ArrayTypeMismatchException</td>
<td>Handles errors generated when type is mismatched with the array type.</td>
</tr>
<tr>
<td>System.NullReferenceException</td>
<td>Handles errors generated from deferencing a null object.</td>
</tr>
<tr>
<td>System.DivideByZeroException</td>
<td>Handles errors generated from dividing a dividend with zero.</td>
</tr>
<tr>
<td>System.InvalidCastException</td>
<td>Handles errors generated during typecasting.</td>
</tr>
<tr>
<td>System.OutOfMemoryException</td>
<td>Handles errors generated from insufficient free memory.</td>
</tr>
<tr>
<td>System.StackOverflowException</td>
<td>Handles errors generated from stack overflow.</td>
</tr>
</tbody>
</table>

**Handling Exceptions**

C# provides a structured solution to the exception handling in the form of try and catch blocks. Using these blocks the core program statements are separated from the error-handling statements.

These error handling blocks are implemented using the **try, catch, and finally** keywords. Following is an example of throwing an exception when dividing by zero condition occurs:

```csharp
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
            {
                .WriteLine("Result: {0}", result);
            }
        }
    }
    static void Main(string[] args)
```
When the above code is compiled and executed, it produces the following result:

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
at ...
Result: 0
```

**Creating User-Defined Exceptions**

You can also define your own exception. User-defined exception classes are derived from the `ApplicationException` class. The following example demonstrates this:

```csharp
using System;
namespace UserDefinedException
{
    class TestTemperature
    {
        static void Main(string[] args)
        {
            Temperature temp = new Temperature();
            try
            {
                temp.showTemp();
            }
            catch(TempIsZeroException e)
            {
                Console.WriteLine("TempIsZeroException: {0}", e.Message);
            }
            Console.ReadKey();
        }
    }

    public class TempIsZeroException: ApplicationException
    {
        public TempIsZeroException(string message): base(message)
        {
        }
    }

    public class Temperature
    {
        int temperature = 0;
        public void showTemp()
        {
            if(temperature == 0)
            {
                throw (new TempIsZeroException("Zero Temperature found"));
            }
            else
            {
                Console.WriteLine("Temperature: {0}", temperature);
            }
        }
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
TempIsZeroException: Zero Temperature found
```
Throwing Objects

You can throw an object if it is either directly or indirectly derived from the `System.Exception` class. You can use a throw statement in the catch block to throw the present object as:

```csharp
Catch(Exception e)
{
    ...
    Throw e
}
```