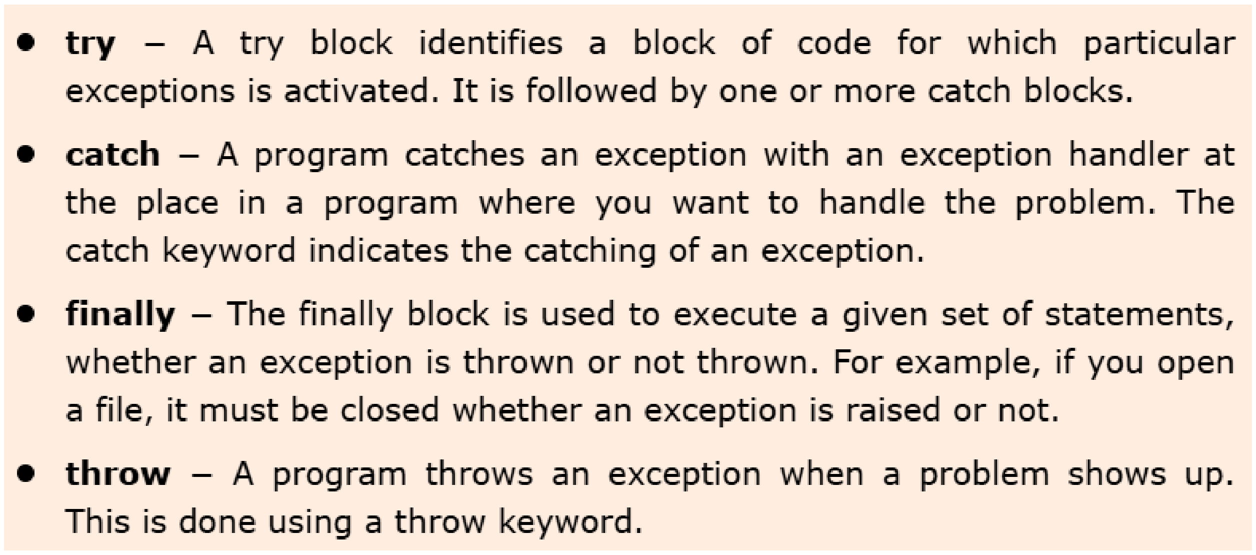
An exception is a problem that arises during the execution of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

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.

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 –

```
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();
      }
   }
}
```

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 **Exception** class. The following example demonstrates this −

```
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: Exception {
   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 –

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

PROGRAMS-PRACTICE

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("Enter a number: ");

            var num = int.parse(Console.ReadLine());

            Console.WriteLine($"Squre of {num} is {num * num}");
        }
        catch
        {
            Console.Write("Error occurred.");
        }
        finally
        {
            Console.Write("Re-try with a different number.");
        }
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("Enter a number: ");

            var num = int.parse(Console.ReadLine());

            Console.WriteLine($"Squre of {num} is {num * num}");
        }
        catch(Exception ex)
        {
            Console.Write("Error info:" + ex.Message);
        }
        finally
        {
            Console.Write("Re-try with a different number.");
        }
    }
}
```

```
static void Main(string[] args)
{
    Console.Write("Please enter a number to divide 100: ");

    try
    {
        int num = int.Parse(Console.ReadLine());

        int result = 100 / num;

        Console.WriteLine("100 / {0} = {1}", num, result);
    }
    catch(DivideByZeroException ex)
    {
        Console.Write("Cannot divide by zero. Please try again.");
    }
    catch(InvalidOperationException ex)
    {
        Console.Write("Invalid operation. Please try again.");
    }
    catch(FormatException  ex)
    {
        Console.Write("Not a valid format. Please try again.");
    }
    catch(Exception  ex)
    {
        Console.Write("Error occurred! Please try again.");
    }
}
```

Example: Nested try-catch

```
static void Main(string[] args)
{
    var divider = 0;

    try
    {
        try
        {
            var result = 100/divider;
        }
        catch
        {
            Console.WriteLine("Inner catch");
        }
    }
    catch
    {
        Console.WriteLine("Outer catch");
    }
}
```

Example: Nested try-catch

```
static void Main(string[] args)
{
    var divider = 0;

    try
    {
        try
        {
            var result = 100/divider;
        }
        catch(NullReferenceException ex)
        {
            Console.WriteLine("Inner catch");
        }
    }
    catch
    {
        Console.WriteLine("Outer catch");
    }
}
```