# Object-oriented programming (OOP) Lecture 7: Exception Handling

## Exception Handling

* It is very common for software applications to get errors and exceptions when executing code.
* If these errors are not handled properly, the application may crash and you may not know the root cause of the problem.
* **Exception handling** is the method of catching and recording these errors in code so you can fix them.
* In C#, the exception handling method is implemented using the **try catch and finally** statement.

## Try, catch, and finally in C#

* The **try**, **catch**, and **finally** statement in C# implements exception handling.
* The **try** encloses the code that might throw an exception.
* The **catch** handles an exception if one exists.
* The **finally** is used for any cleanup work that needs to be done.

try

{

// Statement which can cause an exception.

}

catch (Type x)

{

// Statements for handling the exception

}

finally

{

// Any cleanup code

}

* If any exception occurs inside the try block, the control transfers to the appropriate catch block and later to the finally block.
* The try block can exist either with one or more catch blocks or a finally block or with both catch and finally blocks.
* If there is no exception occurring inside the try block, the control directly transfers to finally block. We can say that **the statements inside the finally block are always executed**. **Note** that it is an error to transfer control out of a finally block by using **break**, **continue**, **return**, or **goto**.

## Uncaught Exceptions in C#

* The following program will compile but will show an error during execution. The **division by zero** is a runtime anomaly, and the program terminates with an error message.
* Any **uncaught exceptions** in the current context propagate to a higher context and look for an appropriate catch block to handle it. If it can't find suitable catch blocks, the default mechanism of the .NET runtime will terminate the execution of the entire program.

class Program

{

static void Main()

{

int x = 0;

int y = 100 / x;

Console.WriteLine("Result = "+y);

}

}

* The modified form of the above program with an exception-handling mechanism is as follows. Here we are using the object of the standard exception class **DivideByZeroException** to handle the exception caused by division by zero.

class Program

{

static void Main()

{

int x = 0;

int y = 0;

try

{

y = 100 / x;

Console.WriteLine("This Line is not Executable if

exception occurs");

}

catch (DivideByZeroException)

{

Console.WriteLine("Can't divide by zero");

}

Console.WriteLine("Result = " + y);

}

}

* In the above case, **the program does not terminate unexpectedly**. Instead, the program control passes from where the exception occurred inside the try block to the catch blocks. If it finds any suitable catch block, it executes the statements inside that catch and continues with the normal execution of the program statements.
* If a finally block is present, the code inside the finally block will get also be executed as shown in the following example.

class Program

{

static void Main()

{

int x = 0;

int y = 0;

try

{

y = 100 / x;

Console.WriteLine("This Line is not Executable if

exception occurs");

}

catch (DivideByZeroException)

{

Console.WriteLine("Can't divide by zero");

}

finally

{

Console.WriteLine("Finally Block");

}

Console.WriteLine("Result = " + y);

}

}

* Both catch and finally blocks are optional. The try block can exist either with **one or more** **catch** or with both catch and finally blocks.

class Program

{

static void Main()

{

int x = 0;

int y = 0;

try

{

y = 100 / x;

Console.WriteLine("This Line is not Executable if

exception occurs");

}

catch (Exception)

{

Console.WriteLine("not valid");

}

finally

{

Console.WriteLine("Finally Block");

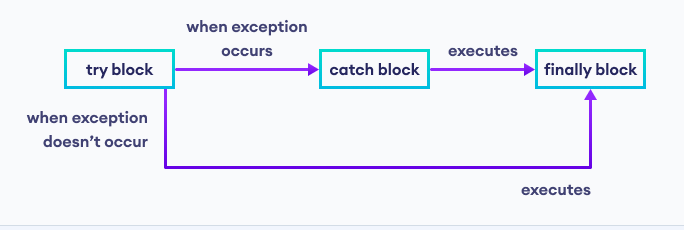
}

Console.WriteLine("Result = " + y);

}

}

* In C#, a **try block must be followed by either a catch or a finally block**. But in above case, since there is no exception handling catch block, **the execution will get terminated**.



## Multiple Catch Blocks

* A try block can throw multiple exceptions, which can be handled using multiple catch blocks. Remember that a **more specialized** catch block should come **before** a **generalized one**. Otherwise, the compiler will show a **compilation error**.

class Program

{

static void Main()

{

int x = 0;

int y = 0;

try

{

y = 100 / x;

Console.WriteLine("This Line is not Executable if

exception occurs");

}

catch (DivideByZeroException)

{

Console.WriteLine("Can't divide by zero");

}

catch (Exception)

{

Console.WriteLine("General Exception");

}

finally

{

Console.WriteLine("Finally Block");

}

Console.WriteLine("Result = " + y);

}

}

## Catching all Exceptions in C#

* By providing a catch block without brackets or arguments, we can catch all exceptions that occurred inside a try block. Even we can use a catch block with an Exception type parameter to catch all exceptions that happen inside the try block since, in C#, all exceptions are directly or indirectly inherited from the Exception class.

class Program

{

static void Main()

{

int x = 0;

int y = 0;

try

{

y = 100 / x;

Console.WriteLine("This Line is not Executable if

exception occurs");

}

catch (DivideByZeroException)

{

Console.WriteLine("Can't divide by zero");

}

catch

{

Console.WriteLine("All Exception");

}

finally

{

Console.WriteLine("Finally Block");

}

Console.WriteLine("Result = " + y);

}

}

## Throwing an Exception in C#

* In the previous section we have seen how to handle exceptions which are automatically raised by CLR. Here, we will see how to raise an exception manually.
* An exception can be raised manually by using the **throw** keyword. Any type of exceptions which is derived from Exception class can be raised using the throw keyword.

class Student

{

public string stdName = "Hani";

}

class Program

{

static void Main()

{

Student std = null;

Program p=new Program();

try

{

p.PrintStdName(std);

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

}

public void PrintStdName(Student s)

{

if(s == null)

{

throw new NullReferenceException("Student object is

null.");

}

Console.WriteLine(s.stdName);

}

}

Another Example

class Program

{

static void Main()

{

int balance = 0;

try

{

balance = 1000000;

checkBalance(balance);

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

}

public static void checkBalance(int b)

{

if(b >999999)

{

throw new Exception("Very High Balance");

}

Console.WriteLine("Balance = "+b);

}

}

## Re-throwing an Exception in C#

* The exceptions we caught inside a catch block can be re-throw to a higher context by using the keyword **throw** inside the catch block. The following program shows how to do this.

class MyClass

{

public void Method()

{

try

{

int x = 0;

int y = 100 / x;

}

catch (DivideByZeroException)

{

throw;

}

}

}

class Program

{

public static void Main()

{

MyClass mc = new MyClass();

try

{

mc.Method();

}

catch (Exception)

{

Console.WriteLine("Exception caught here");

}

Console.WriteLine("LAST STATEMENT");

}

}

## Standard Exceptions in C#

* There are two types of exceptions: **exceptions generated by an executing program** and **exceptions generated by the common language runtime**.
* S**ystem.Exception** is the base class for all exceptions in C#. Several exception classes inherit from this class, including ApplicationException and SystemException. These two classes form the basis for most other runtime exceptions. Other exceptions derive directly from System.Exception includes IOException, WebException, etc.
* The common language runtime (CLR) throws SystemException. The ApplicationException is thrown by a user program rather than the runtime. **We are not recommended** to catch SystemExceptions, nor is it good programming practice to throw SystemExceptions in our applications. The following exceptions are the most common C# exceptions:
* System.OutOfMemoryException
* System.NullReferenceException
* Syste.InvalidCastException
* Syste.ArrayTypeMismatchException
* System.IndexOutOfRangeException
* System.DevideByZeroException

## User-defined Exceptions in C#

* In C#, it is possible to create our exception class. But Exception must be the ultimate base class for all exceptions in C#. So the user-defined exception classes must inherit from either the Exception class or one of its standard-derived classes.

class MyException : Exception

{

public MyException(string message):base(message)

{

}

}

class Program

{

public static void Main()

{

try

{

throw new MyException("User Defined Exception");

}

catch (Exception ex)

{

Console.WriteLine("Exception caught here : "+ ex.Message);

}

Console.WriteLine("LAST STATEMENT");

}

}