Exception Handling

Error: An error is a mistake done by the developer while writing business logic.

Errors are two types than can occur in an application.

- **Syntax errors[compile time errors]**
- **Runtime errors[Exceptions]**

Syntax errors: Occurs when compiling the program due to syntactical mistakes.[keywords misspelled, ; missing etc]

- Compile time errors are detected by compilers.

Runtime errors: Occurs during program execution

- Runtime errors are detected by Runtime System[CLR]
- Runtime errors are called exceptions.

Exception Handling

```
Class syntax_error
{
        Console.WriteLine("Enjoy Errors")
}
```

Semicolon is missing from `Console.WriteLine` statement. This is a syntax error.

Exception Handling

```
class Errors

 {

 int Num1=0;
 int Num2=20;
 int Num3;

 Num3=Num2/Num1;

Console.WriteLine("The Result is {0}", Num3);

 }
```

Division by zero is a run-time error.

Defining Exception

An exception is an abnormal condition that arises while running a program.

Examples:

- Attempt to divide an integer by zero causes an exception to be thrown at run time.
- Attempt to call a method using a reference that is null.
- Attempting to access a file that does not exist on disk.
- Note: When Exception occur at runtime, application terminates un successfully with error message.

Exception handling syntax

- Handling exception that occur at runtime in our application is exception handling.
- Syntax

```
try{
// code that may throw exception
}catch(Exception e){
// handler
}
finally{
//statements
}
```

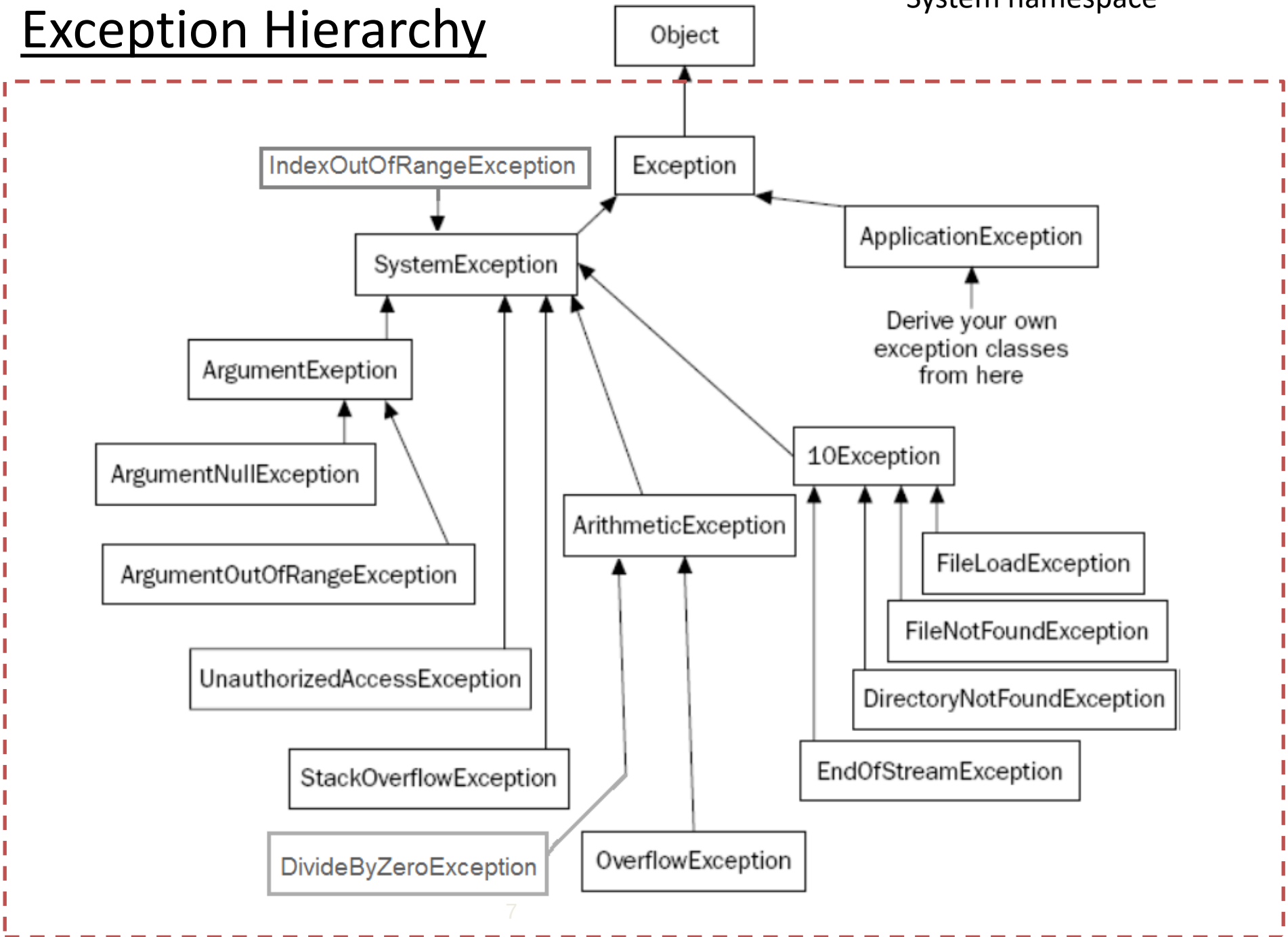
- A try block must either have a catch block or a finally block or both.

Exception Points

- A try block is a block where error are expected to occur.
- The errors are thrown in the form of System.Exception object in C#.
- A catch block will have the code to handle the error. After the catch block executes control goes to the next statement after all the catch blocks in the same level.
- A try block can have one or more catch block where each catch block can handle different types of exceptions. The type of exception that catch block specifies is called exception filter.
- Only one catch block is executed for each exception that is thrown.

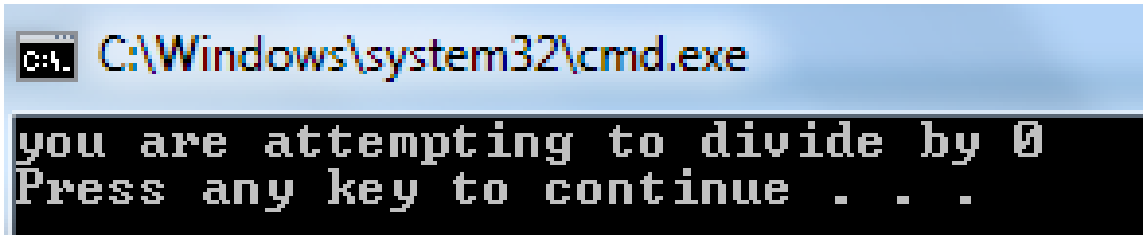
Exception Hierarchy

System namespace



Example: catch handler

```
using System;
class Div{
public static void Main(){
int k=10, j=0;
try{  k=k/j;
Console.WriteLine("hello");
}catch(DivideByZeroException e){
Console.WriteLine("you are attempting to divide by
   0");
}}}
```



Multiple Exception

- Multiple Exception can be handled by using multiple catch.
- The subclass exceptions must be handled before the super class exception.
- The code below handles 2 exceptions:

```
using System;
    class Div{
    public static void Main(string[] s){
        try{
        int k=Int32.Parse(Console.ReadLine());
        j=j/k;
        Console.WriteLine("j="+j);
```

```
}catch(DivideByZeroException d){
 Console.WriteLine("Denominator is 0 " + d.Message); }
catch(FormatException d){
 Console.WriteLine("String conversion into number
                    failed  "+d.Message);}}}
```

Execution Path 1:

Denominator is 0 Attempted to divide by zero.

Execution Path 2:

String conversion into number failed Input string was not in a correct format.

Execution Path 3:

i= 2: **j=5**

Catch-all Exception

- All uncaught exception objects can be handled by a catch-all block which catches **`Exception`** object(the root class of all exception)
- Example:

```
static void Main(string[] s){
      try{
         int k = Int32.Parse(Console.ReadLine());
           j = j / k;
           Console.WriteLine("j=" + j);
           Console.WriteLine(s[1]);
```

```
}catch (DivideByZeroException d){
Console.WriteLine("Denominator is 0 " + d.Message);
} catch(FormatException d){
Console.WriteLine("String conversion into number
failed  "+d.Message);}
catch (Exception d){
Console.WriteLine("Some error occurred "+d);}
}
```

Execution Path:

With no command line argument 2:

j=5

Some error occurred Index was outside the bounds of the array.

Unreachable exception error

```
class Test    {
    static void Main(string[] s){
        try
        {
            int j = 10;
            j = j / s.Length;
            int k = Int32.Parse(s[0]);
            j = j / k;
            Console.WriteLine("j=" + j);
        }
        catch (Exception d){
            Console.WriteLine("Denominator is 0 " + d.Message);
        }
        catch(FormatException d){
            Console.WriteLine("String conversion into number  failed  "+d.Message);}
     /*   catch (Exception d)
        {
            Console.WriteLine(" general error "+d);}
        catch{
            Console.WriteLine("error due to code outside the system");}*/
}}
```

Error List

1 Error | 29 Warnings | 0 Messages

	Description	File
30	A previous catch clause already catches all exceptions of this or of a super type ('System.Exception')	Test.cs

Exception class members

Properties

- **`Message`**
  - Gets a message that describes the current exception.
- **`Source`**
  - Gets or sets the name of the application or the object that causes the error.
- **`StackTrace`**
  - Gets a string representation of the frames on the call stack at the time the current exception was thrown
- **`TargetSite`**
  - Gets the method that throws the current exception.
- **`HelpLink`**
  - Gets or sets a link to the help file associated with this exception

finally

- **try** block can have a **finally** block also apart from the **catch** block.
- **finally** block will execute whether or not an exception occurs.
- It is provided so that the clean up code could be written in all cases whether an error occurs or not, like closing of a file, database connection etc.
- In C#, a **try** block must be followed by either a **catch** or **finally** block .
- Syntax:

```
try{
//Code that may throw exception
}
[catch(SomeException_class1 e) {}
…
 catch(SomeException_classN e) {} ]
[catch(…){}]
[finally{}]
```

Throwing an exception

- It is possible to throw an exception explicitly from code.
- This is done using **`throw`** keyword.
- Syntax:

```
throw excepobject;
```

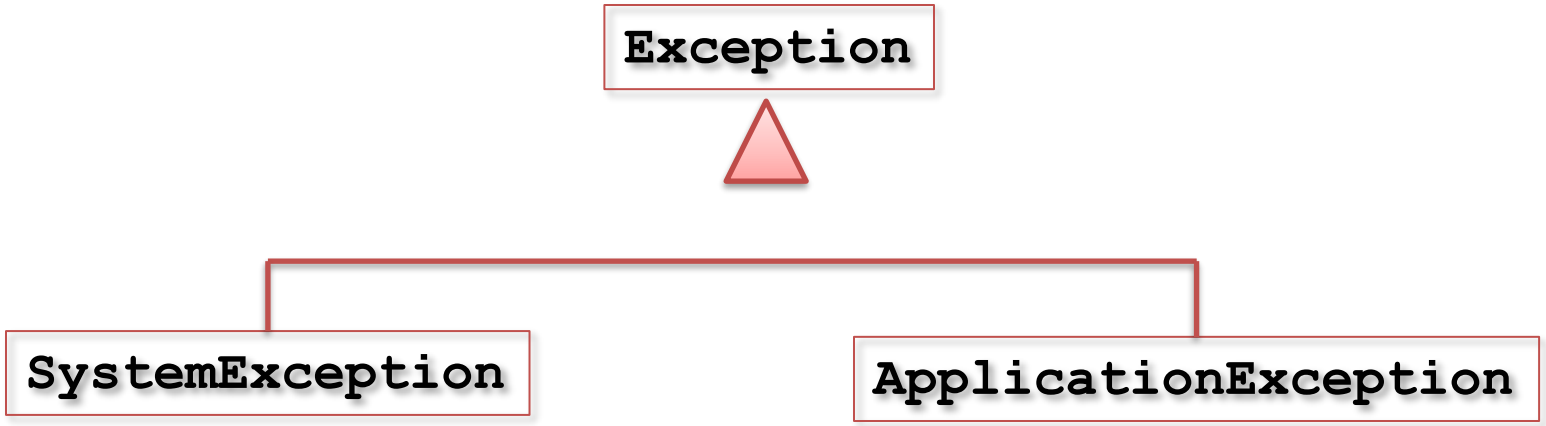
- Example:

```
using System;
  class Test    {
      static void Main(string[] s){
      double d1=10,d2=s.Length;
      if (d2 == 0) throw new DivideByZeroException();
      d1 = d1 / d2;  }}
```

```
Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
   at Test.Main(String[] s) in C:\Users\deepa.krishnan\Documents\Visual Studio 2
008\Projects\Chapter1Ex1\Chapter1Ex1\Test.cs:line 5
```

Types of exception

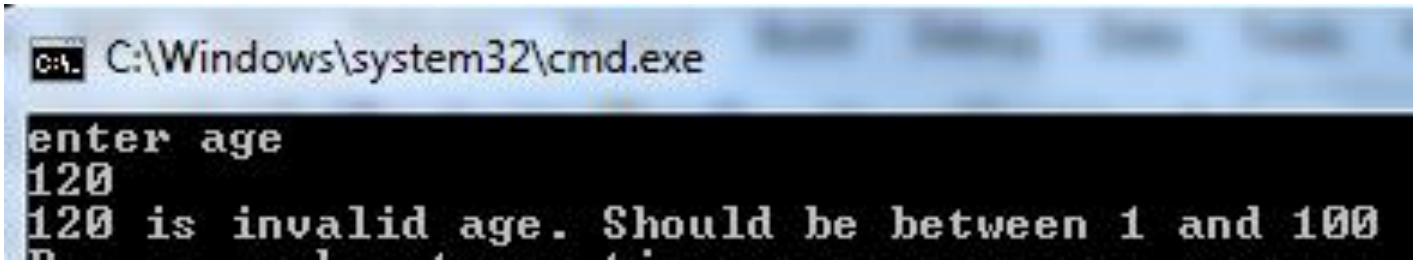
- Standard Exception
  - Exception thrown by the CLR
  - CLR throws objects of type **`SystemException`**
- Application Exception
  - thrown by a user program rather than the runtime.
  - Inherits from **`ApplicationException`**



Application exception

```
using System;
class AgeException : ApplicationException{
string s;
 public AgeException(string str) {
  s=str +" is invalid age. Should be between 1 and
  100";
 }
public override string ToString(){
return s;
}
}
```

```
class Test{
 public static void Main() {
  try  {
   Console.WriteLine("enter age");
   string s=Console.ReadLine();
   int num = Int32.Parse(s);
      if(num<1 || num>100)
            throw new AgeException(s);
  }
  catch(AgeException  e) {
            Console.WriteLine (e);
  }
  }
}
```



What will happen if you enter an *alphabets instead of number for age?*

List of popular exceptions

- Exceptions that we have already seen
  - **`SystemException, NullReferenceException, DivideByZeroException, TypeInitializationException, ArgumentException, FormatException, IndexOutOfRangeException, InvalidCastException`**
- Other popular exceptions
  - **`ArithmeticException:`** exceptions that occur during arithmetic operations, base class for **`DivideByZeroException`** and **`OverflowException`**
  - **`OverflowException:`** Thrown when an arithmetic operation in a checked context overflows.

```
byte b = 255;  checked { b++; }
```

Test your understanding

Will the set of code listed below throw any exception? If so what exception will be thrown. Which line will throw the exception?

```
1. class A { }
   class B : A { }
   class C :A{ }
   class Test{
       static void Main(string[] s)     {
           A a = new B();
           C c = (C)a;
       }
   }

2. using System;
   class Test{
       static void Main(string[] s)    {
   Console.Write(s[0]);
   Console.Write(s.Length);     }}
```

3.
```
using System;

class Test{
    static Test(){
    throw new Exception();
    }
    static void Main(string[] s) { }
}
```

4.
```
using System;

class Test{
    static void Main(string[] s)    {
   int b = 1000000000;
         b = b * b;
         Console.Write(b);
      }
}
```

5.
```
using System;

class Test{
  static  string str;
    static void Main(string[] s)   {
                    Console.Write(str.Length);
         }
}
```

6.
```
class Test{
        static void Main()   {
        for (; ; ) Main();
         }
}
```

7.
```
class Test{
     static void Main()   {
         string[] s = { "abc", "acc", "dd" };
         object[] k = (object[])s;
         k[0] = 1;

   }}
```