BUILT-IN USER DEFINED AND CHAINED EXCEPTIONS

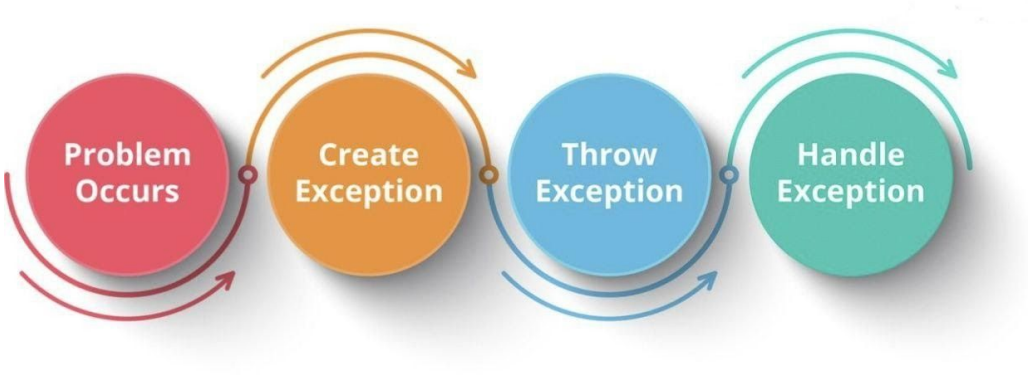
Exception:

An exception is an event that disrupts the normal flow of the program.Exception Handling is a mechanism to handle runtime error.Throwable is the base class for Exception (An exception is a sudden and an unexpected event which leads to the sudden termination of the program during execution of a program (i.e) at the runtime of a program.)

Types of exception:

- Checked Exception
- Unchecked Exception

The ways to handle exception:



Difference between Checked & Unchecked Exception

Checked	Unchecked
• An exception that is checked by the compiler at compilation-time • These exceptions cannot simply be ignored,the programmer should handle these exception • eg.,FileNotFound Exception,IO Exception,Etc.,	• An exception that occurs at the time of execution • These are also called as Runtime Exception • Runtime exceptions are ignored at the time of compilation

	• eg.,Arithmetic Exception,ArrayIndexOutOfBound Exception ,Etc.,

Syntax of Exception:

```
class Exception
{
        Public static void main(string args[])
        {
            Try
            {
                //code that may raise exceptions
            }
            catch(Exception e)
            {
                // rest of the program
            }
        }
}
```

In this we are going to discuss…..

- ❖ **Built-in Exceptions**
- ❖ **User defined exceptions**
- ❖ **Chained exceptions**

Built-in Exceptions:

Built-in Exceptions are the exceptions which are available in java libraries.These exceptions are suitable to explain certain error situations.

Examples of Built-in Exception:

- Arithmetic Exception
- ArrayIndexoutofBound Exception
- ClassNotFound Exception

- FileNotFound Exception
- IO Exception(Input/Output)
- Interrupted Exception
- NoSuchMethod Exception
- NullPointer Exception
- NumberFormat Exception
- StringIndexOutOfBound Exception

Arithmetic Exception:

Arithmetic Exception is thrown when an exceptional condition has occurred in an arithmetic operation

Example:

```
class ArithmeticException
{
        public static void main(string args[])
        {
            Try
            {
                int a=10,b=0;
                Int c=a/b;
                System.out.println("a+b="+c);
            }
            catch(ArithmeticException e)
            {
                System.out.println("Can't Divide a Number by zero");
            }
        }
}
```

Output:

Can't Divide a Number by zero

ArrayIndexOutOfBounds Exception:

ArrayIndexOutOfBounds Exception is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Example:

```
class ArrayIndexOutOfBound_Demo
{
     public static void main(String args[])
     {
          try
          {
               int a[] = new int[5];
               a[6] = 9;                              // accessing 7th element in an array
           }                                          // size 5

          catch (ArrayIndexOutOfBoundsException e)
          {
                System.out.println("Array Index is Out Of Bounds Exception");
          }
     }
}
```

Output:

Array Index is Out Of Bounds Exception

NullPointer Exception:

This exception is raised when referring to the members of a null object. Null represents nothing

Example:

```
class NullPointer
{
```

```
        Public static void main(string args[])
        {
            try
            {
                    string str=NULL;
                    System.out.println(str.length());
            }
           Catch(NullPointerException e)
           {
                    System.out.println("Null Pointer Exception");
            }
        }
}
```

Output:

Null Pointer Exception

NumberFormat Exception:

This exception is raised when a method could not convert a string into a numeric format.

Example:

```
class NumberFormat
{
        Public static void main(string args[])
        {
            try
            {
                    int num=Integer.parseInt("Java");
                    System.out.println(num);
            }
            catch(NumberFormatException e)
            {
                   System.out.println("Number Format Exception")
            }
       }
```

```
}
```

Output:

Number Format Exception

ClassCastException:

ClassCastException in java is a run time error that occurs when an object can not be casted to another type.

Example:

```
class Main
 {
       public static void main(String[] args)
       {
            try
            {
                 String s = new String("Java");
                 Object o = (Object)s;
                 Object o1 = new Object();
                 String s1 = (String)o1;
             }
             catch(ClassCastException e)
             {
                  System.out.println("Class Cast Exception")
              }
       }
}
```

Output:

Class Cast Exception

StringIndexOutOfBounds Exception:

It is thrown by String class methods to indicate that an index is either negative than the size of the string.

Example:

```
class StringIndexOutOfBound
{
	public static void main(String args[])
	{
		try
		{
			String a = "This is like chipping ";    // length is 22
			char c = a.charAt(24);                  // accessing 25th element
			System.out.println(c);
		}
		catch (StringIndexOutOfBoundsException e)
		{
			System.out.println("String Index Out Of Bounds Exception");
		}
	}
}
```

Output:

```
String Index Out Of Bounds Exception
```

Interrupted Exception:

It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.

Example:

```
Class sample extend thread
{
```

```
  Public static void main(string args[])
{
 Sample s=new sample() ;
 s.start() ;
 s.interrupt() ;
}
Public void run()
{
 try
{
Thread. Sleep(100);
}
Catch(Exception e)
{
System.out.println(e);
}
}
}

Output:
java.lang.interruptedException: sleep interrupted
```

FileNotFound Exception:

This Exception is raised when a file is not accessible or does not open

Example:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;

class Main
 {
      public static void main(String args[])
      {
```

```
            try
            {
                File file = new File("E:// file.txt");     // Following file does not exist

                FileReader fr = new FileReader(file);
            }
            catch (FileNotFoundException e)
            {
                  System.out.println("File does not exist");
             }
        }
}
```

Output:

File does not exist

ClassNotFound Exception:

This Exception is raised when we try to access a class whose definition is not found.

Example:

```
Public class Main
{
       Public static void main(String args[])
       {
            try
          {
                Class.forName(“Java”);
           }
          catch(ClassNotFoundException e)
          {
                System.out.println(“classnotfound exception”)
           }
     }
}
```

Output:

classnotfound exception

User defined exception:

- In java we have already defined, exception classes such as ArithmeticException, NullPointerException etc. These exceptions are already set to trigger on pre-defined conditions such as when you divide a number by zero it triggers ArithmeticException.
- Followed by that we can create our own exception class in java and throw that exception using **throw** keyword. These exceptions are known as **user-defined** or **custom** exceptions.

throw keyword:

- The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.
- We can throw either **checked or unchecked exceptions.**
- The throw keyword is mainly used to throw custom exceptions.

Syntax:

```
throw new exception_class("error message");
```

For example:

➔ `throw new ArithmeticException("divisible by zero(0)");`

A simple example program for user defined exception using throw keyword:

```
import java.util.Scanner;
class LowMarkException extends Exception
{
    public LowMarkException ()
    {
        System.out.println ("The student is failed");
    }
}
class Main
{
```

```
    public static void main (String[]args)
  {
     try
     {
          Scanner s = new Scanner (System.in);
          int mark;
          System.out.println ("Enter the mark of student (0-100) ....");
          mark = s.nextInt ();
          if(mark>=50 && mark <=100)
          {
               System.out.println ("The student is Passed");
               System.out.println ("Exception not caught");
          }
          else
          {
              throw new LowMarkException ();   //throwing user defined exception
          }
      }

     catch (LowMarkException e)
     {
           System.out.println ("Exception caught");
     }
   }
}
```

Output:1 // if input is < 50

```
Enter the mark of student...45
The student is Failed
Exception caught
```

Output:2 // if input is >=50

```
Enter the mark of student...89
The student is Passed
Exception not caught
```

Notes:

- ❖ User-defined exceptions must extend Exception class.
- ❖ The exception is thrown using the throw keyword.

Explanation:

In the above example,we need to check whether the student got a pass or fail mark.Inside the try block we getting the input from the user and check the condition. If the condition is false the exception is thrown by the throw keyword to the created exception class by extending the Exception class (catch block).

Advantage of user defined or custom exception:

- ★ **Custom / User defined exceptions** provide you the flexibility to add attributes and methods that are not part of a standard **Java exception**.
- ★ If you reuse an existing exception, any piece of your code that catches the exception has to deal with the possibility that the actual exception wasn't thrown by your code, but by some other library party code.
- ★ This tends to make error handling **more flakey.**

Chained Exceptions:

Chained Exceptions allows you to relate one exception with another exception, i.e one exception describes the cause of another exception.

Constructors Of Throwable class Which support chained exceptions in java :

- Throwable(Throwable cause) - the cause is the current exception.
- Throwable(String msg, Throwable cause) - msg is the exception message, the cause is the current exception.

Methods Of Throwable class Which support chained exceptions in java :

- getCause() - method returns the actual cause associated with the current exception.

- initCause() (Throwable cause) - set an underlying cause(exception) with invoking exception.

Example : 1

```
class Main
{
    public static void main(String[] args)
    {
        try
        {
            NumberFormatException a = new NumberFormatException("=>Exception");
            throw a;
        }

         catch(NumberFormatException a)
         {
              System.out.println(a);
         }
    }
}
```

Output:

```
java.lang.NumberFormatException: =>Exception
```

Explanation:

- Here NumberFormatException was thrown but the actual cause of exception was a Nullpointer exception.
- The method will throw only a NumberFormatException to the user. So the user would not come to know about the actual cause of exception(Nullpointer exception).

Example : 2 (By using initCause(Throwable cause) in try block and getcause() in catch block we are able to display the actual error occurred in it to the user)

```
class Main
{
    public static void main(String[] args)
    {
        try
        {
            NumberFormatException a = new NumberFormatException("=> Exception");

            a.initCause(new NullPointerException("=> Actual cause of the Exception"));

            throw a;
        }
        catch(NumberFormatException a)
        {
            System.out.println(a);
            System.out.println(a.getCause());
        }
    }
}
```

Output :

```
java.lang.NumberFormatException: =>Exception
java.lang.NullPointerException: =>Actual cause of the Exception
```

Explanation:

- Here NumberFormatException was thrown but the actual cause of exception was a NullPointerException.
- The method will throw only a NumberFormatException to the user. So that by using initCause(Throwable cause) in try block and getcause() in catch block) users would come to know about the actual cause of exception(NullPointerException) by displaying it .

Note:

If you didn't give initCause(Throwable cause) in try block , the above

program will give the output like this...

Output :

```
java.lang.NumberFormatException: =>Exception
null
```