Exception Handling

1. Introduction
2. Runtime Stack Mechanism
3. Default Exception Handling in Java
4. Exception Hirarchy
5. Customized Exception Handling By using try-catch
6. Control flow in try-catch
7. Methods to Print Exception Information
8. Try with multiple catch blocks
9. finally block
10. Difference between final,finally,finalize
11. Control Flow in try-catch-finally
12. Control Flow in Nested try-catch-finally
13. various possible combinations of try-catch-finally
14. throw keyword
15. throws keyword
16. Exception Handling Keywords summary
17. Various possible compile time errors in Exception Handling.
18. Top-IOExceptions
19. 1.7 Version Enhancements
    a. 1.try-with Resources
    b. 2.Multi-catch block.

Introduction:

Exception:

Default Exception Handling:

Inside a Method ifany exception occurs the method in which it is raised is responsible to create exception object by including the following information.

i.name of exception
ii.description of exception
iii.Location at which exception occurs(Stacke Trace)

--After creating exception object,method handover that object to the JVM.

--JVM will check whether the method contains any exception handling code or not.if method does not contain exception handling code then JVM Terminates that method abnormally and removes corresponding entry from the stack.

--Then JVM identifies caller method & Checks whether caller method contains any handling code or not.

--ifthe caller method does not conatin handling code then JVM terminates that caller method also abnormally and removes corresponding entry from the stack.

--This process will be continued until main() method and ifthe main() also does not contain handling code then JVM terminates main() also abnormally & removes corresponding entry from the stack.

--Then JVM hand overs responsibility of exception handling to default exception handler,which is the part of the JVM.

--default exception handler prints exception information in the follwing format & terminates program abnormally.

Syntax:

Exception in thread XXXXXX name of Exception:Description

Stack Trace

Ex 1:

```
class Test
{
	public static void main(String[] args)
	{
		doStuff();
	}

	public static void doStuff()
	{
		doMoreStuff();
	}
	public static void doMoreStuff()
```

```
	{
		//System.out.println("Hello");
		System.out.println(10/0);
	}
}
```

```
D:\Yellaswamy_ClassNotes\UNIT-3\Ex1>javac Test.java
D:\Yellaswamy_ClassNotes\UNIT-3\Ex1>java Test
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at Test.doMoreStuff(Test.java:15)
	at Test.doStuff(Test.java:10)
	at Test.main(Test.java:5)

D:\Yellaswamy_ClassNotes\UNIT-3\Ex1>
```

Ex2:

```
class Test
{
	public static void main(String[] args)
	{
		doStuff();
		System.out.println(10/0);
	}

	public static void doStuff()
	{
		doMoreStuff();
		System.out.println("Hello");
	}
	public static void doMoreStuff()
	{
		System.out.println("Hi");

	}
}
```

output:

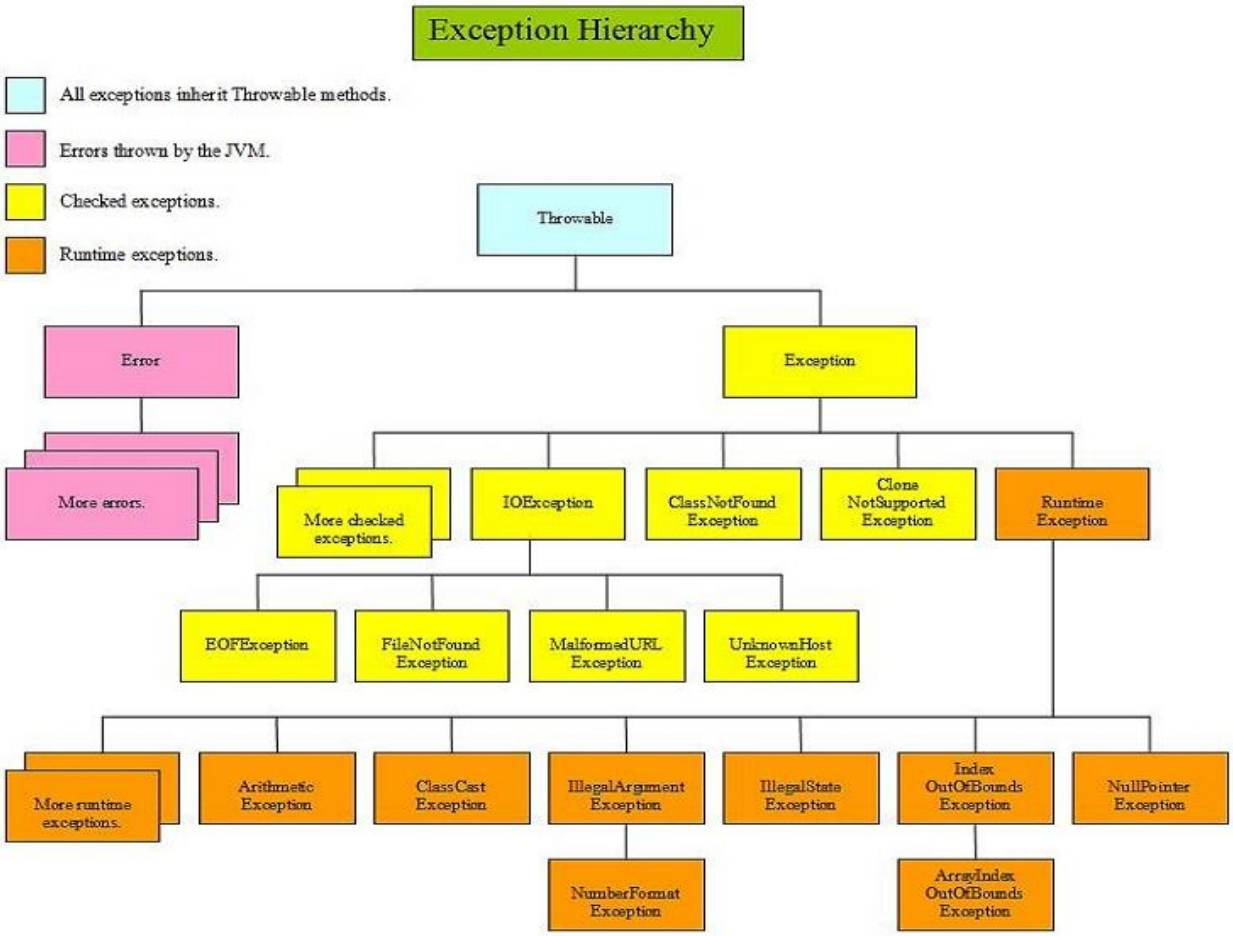
```
D:\Yellaswamy_ClassNotes\UNIT-3\Ex1>java Test
Hi
Hello
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Test.main(Test.java:6)

D:\Yellaswamy_ClassNotes\UNIT-3\Ex1>
```

Note:

in a program at least one method terminates abnormally then the program termination is abnormal termination.

--If all methods terminated normally then only program termination is normal termination.

Exception Hierarchy:



Throwable class acts as root for java exception hirarchy.

Throwable class defines 2 child classes
Throwable

Exception
Error

Exception:
Most of the times exceptions are caused by our program & these are recovarable.

Ex:

If our programming reuirement is Read Remote File locating at USA then we will getFileNotFoundException if FileNotFoundException occurs we can provide local file & continue rest of the program normally.

```
try
{
        Read data from remote file locating at USA
}
catch (FileNotFoundException e)
{
        use local file & continue
                rest of the program normally
}
```

Error:
Most of the time Errors are not caused by our programs and these are lack of System Resources.

--Errors are non recovarable.

Ex:

IfOutofMemoryError occurs being a programmer we can't do anything and the program will be terminated abnormally.

System Admin or Server Admin is responsible to increase Heap Memory.

Ex:

```
import java.io.*;
class MyFileEx
{
public static void main(String[] args)
{
PrintWriter pw=new PrintWriter("Hello.txt");
pw.println("CMRCET");
}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\Ex2>javac MyFileEx.java
MyFileEx.java:6: unreported exception java.io.FileNotFoundException; must be cau
ght or declared to be thrown
PrintWriter pw=new PrintWriter("Hello.txt");
          ^
1 error

D:\Yellaswamy_ClassNotes\UNIT-3\Ex2>
```

Checked vs Unchecked Exception:

Checked Exception:

The Exceptions which are checked by compiler for smooth excecution of the program are called checked exceptions.

Ex:

HallTicketMissingException

PenNotWritingException

FileNotFoundException

In our program ifthere is a chance of raising checked exception compulsory we should handle that checked exception(either by try-catch or by throws) keyworsds otherwise we willgetcompile time error.

```
import java.io.*;
class Test
{
public static void main(String[] args)
{
PrintWriter out=new PrintWriter("abc.txt");
out.println("Hello");
}
```

```
}
```

output:

```
D:\Yellaswamy_ClassNotes>cd UNIT-3

D:\Yellaswamy_ClassNotes\UNIT-3>javac Test.java
Test.java:6: unreported exception java.io.FileNotFoundException; must be caught
or declared to be thrown
PrintWriter out=new PrintWriter("abc.txt");

            ^
1 error

D:\Yellaswamy_ClassNotes\UNIT-3>
```

Ex2:

```
import java.io.*;
class Test
{
public static void main(String[] args)throws FileNotFoundException
{
PrintWriter out=new PrintWriter("abc.txt");
out.println("Hello");
System.out.println(10/0);
}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3>javac Test.java

D:\Yellaswamy_ClassNotes\UNIT-3>java Test
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at Test.main(Test.java:8)

D:\Yellaswamy_ClassNotes\UNIT-3>
```

Methods to Print exeception information:

Ex:1 printStackTrace() Method

```
import java.io.*;
class Test
{
public static void main(String[] args)throws FileNotFoundException
{
try
{
        System.out.println(10/0);
}
catch (ArithmeticException e)
{
        e.printStackTrace();
}

}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3>javac Test.java

D:\Yellaswamy_ClassNotes\UNIT-3>java Test
java.lang.ArithmeticException: / by zero
    at Test.main(Test.java:8)

D:\Yellaswamy_ClassNotes\UNIT-3>
```

Ex 2:toString() Method

```
import java.io.*;
class Test
{
public static void main(String[] args)throws FileNotFoundException
{
try
{
        System.out.println(10/0);
}
catch (ArithmeticException e)
{
```

```
        //e.printStackTrace();
        System.out.println(e);
        System.out.println(e.toString());
}

}
}
```
output:

```
D:\Yellaswamy_ClassNotes\UNIT-3>javac Test.java

D:\Yellaswamy_ClassNotes\UNIT-3>java Test
java.lang.ArithmeticException: / by zero
java.lang.ArithmeticException: / by zero

D:\Yellaswamy_ClassNotes\UNIT-3>
```

Ex 3:getMessage() method

```
import java.io.*;
class Test
{
public static void main(String[] args)throws FileNotFoundException
{
try
{
        System.out.println(10/0);
}
catch (ArithmeticException e)
{
        //e.printStackTrace();
        //System.out.println(e);
        //System.out.println(e.toString());
        System.out.println(e.getMessage());

}

}
}
```
output:

```
D:\Yellaswamy_ClassNotes\UNIT-3>javac Test.java

D:\Yellaswamy_ClassNotes\UNIT-3>java Test
/ by zero

D:\Yellaswamy_ClassNotes\UNIT-3>
```

Customized Exception Handling By using try-catch

without try-catch

==========

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("statement1");
		System.out.println(10/0);
		System.out.println("statement3");
	}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\IIDCheckedException\Test>javac Test.java

D:\Yellaswamy_ClassNotes\UNIT-3\IIDCheckedException\Test>java Test
statement1
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at Test.main(Test.java:6)

D:\Yellaswamy_ClassNotes\UNIT-3\IIDCheckedException\Test>
```

Abnormal Termination this is not recomanded approach.

With try-catch

==========

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

```
	{
		System.out.println("statement1");
		try
		{
			System.out.println(10/0);

		}
		catch (ArithmeticException e)
		{
			System.out.println(10/2);
		}

		System.out.println("statement3");
	}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\IIDCheckedException\Test>javac Test.java

D:\Yellaswamy_ClassNotes\UNIT-3\IIDCheckedException\Test>java Test
statement1
5
statement3

D:\Yellaswamy_ClassNotes\UNIT-3\IIDCheckedException\Test>
```

Normal Termination this reomanded approach

Syntax:

```
try
{
	Risky code
}
catch (Exception e)
{
	Handling code
}
```

control flow in try-catch:

```
=======================
try
{
        stmt1;
        stmt2;
        stmt3;
}
catch (Exception e)
{
        stmt4;
}
stmt 5;
```

case 1:
ifthere is no exception 1,2,3,5 statements executed.normal termination.

case2:
if -exception raised at stmt2 and corresponding catch-block matched
1,4,5
Normal Termination

case3:
if -exception raised at stmt2 and corresponding catch-block is not matched

1.
Abnormal termination
case4:
if -exception raised at stmt4 or stmt 5 then it is always abnormal termination.

Methods to Print Exception Information:
==========

1.public java.lang.String getMessage();
2.public java.lang.String toString();
3.public void printStackTrace();

D:\Yellaswamy_ClassNotes\UNIT-3\IIC>javac Test.java

```
D:\Yellaswamy_ClassNotes\UNIT-3\IIC>java Test
statement1
5
statement3

D:\Yellaswamy_ClassNotes\UNIT-3\IIC>javap java.lang.Throwable
Compiled from "Throwable.java"
public class java.lang.Throwable extends java.lang.Object implements java.io.Ser
ializable{
   public java.lang.Throwable();
   public java.lang.Throwable(java.lang.String);
   public java.lang.Throwable(java.lang.String, java.lang.Throwable);
   public java.lang.Throwable(java.lang.Throwable);
   public java.lang.String getMessage();
   public java.lang.String getLocalizedMessage();
   public java.lang.Throwable getCause();
   public synchronized java.lang.Throwable initCause(java.lang.Throwable);
   public java.lang.String toString();
   public void printStackTrace();
   public void printStackTrace(java.io.PrintStream);
   public void printStackTrace(java.io.PrintWriter);
   public synchronized native java.lang.Throwable fillInStackTrace();
   public java.lang.StackTraceElement[] getStackTrace();
   public void setStackTrace(java.lang.StackTraceElement[]);
}


D:\Yellaswamy_ClassNotes\UNIT-3\IIC>
```

---

try-with multiple catch blocks:

==========

Ex1:

```
try
{
        Risky code
}
catch (Exception e)
```

```
{
        //Any type Exception
}
```

Worst Practice.

Ex2:

```
try
{
        //Risky code
}
catch (ArithmeticException e)
{
        //perform alternate arithmetic operations
}
catch(SQLException e)
{
        //use mysql db instead of oracle db
}
catch(FileNotFoundException e)
{
        //use local file instead of remote file

}

catch(Exception e)
{
        //default exception handling

}
```

Best Practice

LoopHoles:

try-with multiple catch blocks present then the order of catch blocks is very important we have take child first then parent.otherwise we will get -compile time error.

Exception XXX has been already caught.

Ex1:

```
try
{
        //risky code
}
catch (Exception e)
{
}
catch(ArithmeticException e)
{
}
```

CE:exception java.lang. AE has been caught.

Program:

```
class Test
{
public static void main(String[] args)
{
        try
        {
                System.out.println(10/0);
        }
        catch (Exception e)
        {
        }
        catch (ArithmeticException e)
        {
        }


}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\Ex3>javac Test.java
Test.java:12: exception java.lang.ArithmeticException has already been caught
    catch (ArithmeticException e)
    ^
1 error
```

D:\Yellaswamy_ClassNotes\UNIT-3\Ex3>

Ex2:

```
try
{
        //risky code
}
catch (ArithmeticException e)
{
}
catch(Exception e)
{
}
```

Program:

```
class Test
{
public static void main(String[] args)
{
        try
        {
                System.out.println(10/0);
        }
        catch (ArithmeticException e)
        {
                System.out.println(e.toString());
        }
        catch (Exception e)
        {
                System.out.println("Hello");
        }


}
}
```

output:

D:\Yellaswamy_ClassNotes\UNIT-3\Ex3\case2>javac Test.java

```
D:\Yellaswamy_ClassNotes\UNIT-3\Ex3\case2>java Test
java.lang.ArithmeticException: / by zero

D:\Yellaswamy_ClassNotes\UNIT-3\Ex3\case2>
```

Ex3:

```
try
{
        risky code
}
catch (ArithmeticException e)
{
}
catch (ArithmeticException e)
{
}
```

CE:We can't declare two catch blocks for the same exception otherwise we will get compile time error.

==========

Diffrence between final,finally,finallize():

==========

final:

--final is the modifier applicable for classes and methods and variables.

if class declared as final we can't extend that class that is we can't create child class,that is inheritance is not possible forfinal classes

ifa method is final then we can't override that method in the child class.

ifa variable declared as final then we can't perform reassignment for that variable.

finally:

--is a block always associated with try-catch to maintain clean up code.

Ex:

```
try
{
        //Risky code
}
catch (Exception e)
```

```
{
        //Handling code
}
finally
{
        //clean up code
}
```

The speciality of finally block is it will be excecuted always irrespective of whther exception raised or not raised or handled or not.
---

finalize():
---
finalize() is a method always invoked by garbage collector just before destroy an object to perform clean up activities.

once finalize() method completes garbage collector destroy that object.

Note:
finally block is responsible to perform clean up activities related to try
block that is what ever resources we opened as the part of try
block will be closed inside finally block.

where as finalize() method is responsible to perform clean up activities associated with object.

various possible combinations of try-catch-finally:
==========
1.
```
try
{

}
catch ()
{
}
```

2.
```
try
{
```

```
}
catch (X e)
{
}
catch (Y e)
{
}
```

3.

```
try
{

}
catch (X e)
{
}
catch (X e)
{
}
```
CE:Exception X has already been caught

4.
```
try
{

}
catch (X e)
{
}
finally
{
}
```

5.
```
try
{

}
finally
```

```
{
}
```

6.try
```
{

}
catch (X e)
{
}
try
{

}
catch (Y e)
{
}
```

7.try
```
{

}
catch (X e)
{
}
try
{

}
finally
{
}
```

8.try
```
{

}
```
CE:trywithout catch or finally

9.
```
catch(X e)
```

{
}
CE:catch with out try

10.
finally
{
}
CE:
finallywithout try

11.
try
{

}
finally
{
}
catch(x e)
{
}
CE:catch without try

12.
try
{

}
Sop("hello");
catch (X e)
{
}
CE:1
trywithout catch or finally
CE 2:catch with out try-

13.
try
{

```
}
catch (X e)
{
}
Sop("hello");
catch(Y e)
{
}
CE:catch without try-
-------------------------
14.
try
{

}
catch (X e)
{
}
sop("Hello");
finally
{
}
CE:finally without try-
----------------------
-------------------------
----------------------
15.
try
{
        try
        {

        }
        catch (X e)
        {
        }
}
catch (X e)
{
}
---------------------------
```

16.

```
try
{
        try
        {

        }

}
catch (X e)
{
}
```

CE:
try-without catch or finally

17.

```
try
{
        try
        {

        }
        finally
        {
        }
}
catch (X e)
{
}
```

18.

```
try
{

}
catch (X e)
{
        try
        {
```

```
        }
        finally
        {
        }
}
```

19.
```
try
{

}
catch (X e)
{
        finally
        {
        }
}
```
CE:finally without try-

20.
```
try
{

}
catch (X e)
{
}
finally
{
        try
        {

        }
        catch (X e)
        {
        }
}
```

21.
```
try
```

```
{

}
catch (X e)
{
}
finally
{
        finally
        {
        }
}

CE:finally without try-
----------------------------
22.
try
{

}
catch (X e)
{
}
finally
{
}
finally
{
}
CE:
finally without try-
-------------------------
23.
try
Sop("try");
catch (X e)
{
        Sop("catch");
}
finally
{
```

```
}
//invalid
-------------------------
24
try
{

}
catch (X e)
sop("catch");
finally
{
}
//invalid
----------------------
25.
try
{

}
catch (X e)
{
}
finally
sop("hello");
//invalid
-----------------------------------
```

--In try-catch-finally order is important.
--when ever we are writing try-compusory we should write either catch or finally otherwise we will get-compile time error that is try-without-catch is invalid.
--when ever we are writing catch block compulsory try-block must required.That is catch-without try-is invalid.
--when ever we are writing finally block compulosy we should write try-block.that is finally with out try-is invalid.
--inside try-catch and finally block we can declare try-catch-finally blocks.That is nesting of try-catch-finally is allowed.
--for-try-catch-finally blocks curly braces are mandatory.

---
throw keyword:

case1:

```
class Test
{
public static void main(String[] args)
{
        System.out.println(10/0);


}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\throw>java Test
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Test.main(Test.java:5)

D:\Yellaswamy_ClassNotes\UNIT-3\throw>
```

---
case2:

```
class Test
{
public static void main(String[] args)
{
        throw new ArithmeticException("/ by Zero Explicitly");


}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\throw>javac Test.java

D:\Yellaswamy_ClassNotes\UNIT-3\throw>java Test
Exception in thread "main" java.lang.ArithmeticException: / by Zero Explicitly
    at Test.main(Test.java:6)

D:\Yellaswamy_ClassNotes\UNIT-3\throw>
```

```
//case1
class Case1
{
//static ArithmeticException e=new ArithmeticException();
static ArithmeticException e;
public static void main(String[] args)
        {
        throw e;
        }
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>javac Case1.java

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>java Case1
Exception in thread "main" java.lang.NullPointerException
    at Case1.main(Case1.java:9)

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>
```

---

case 2:

```
class Test
{
public static void main(String[] args)
{
        System.out.println(10/0);
        System.out.println("Hello");
        //throw new ArithmeticException("/by zero explicitly");



}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>javac Test.java

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>java Test
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at Test.main(Test.java:5)

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>
----------------------
class Test
{
public static void main(String[] args)
{
		//System.out.println(10/0);
		//System.out.println("Hello");
		throw new ArithmeticException("/by zero explicitly");
		System.out.println("Hello");




}
}
output:

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>javac Test.java
Test.java:8: unreachable statement
	System.out.println("Hello");
	^
1 error

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>
---------------
Case3:
class Case3
{
public static void main(String[] args)
{
throw new Case3();
}
}

output:

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>javac Case3.java
Case3.java:5: incompatible types
```

```
found   : Case3
required: java.lang.Throwable
throw new Case3();
    ^
1 error

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>
```

```
class Case3 extends RuntimeException
{
public static void main(String[] args)
{
throw new Case3();
}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>javac Case3.java

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>java Case3
Exception in thread "main" Case3
      at Case3.main(Case3.java:5)

D:\Yellaswamy_ClassNotes\UNIT-3\throw\IICthrow>
```

---

various possible compile time errors in exception handling:

==========

1.unreported exception xxx ;must be caught or declared to be thrown
2.Exception xxx has already been caught
3.Exception xxx is never thrown in body of corresponding try,statement
4.unreachable statemet
5.incomptible types
6.try,without catch or finally
7.catch without try.

8.finally without try.

Top-10 Exceptions:
==========

In JAVA 3 things will raise an exceptions

1.JVM Raise an Exceptions
Ex:RE:AE
2.Explicitly raised by Programmer
Programatic Exceptions
3.API Developer
Ex:

```
class Test
{
public static void main(String args[])
{
Thread t=new Thread();
t.setPriority(5);
t.setPriority(15);
}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3\Top10Exceptions>javac Test.java

D:\Yellaswamy_ClassNotes\UNIT-3\Top10Exceptions>java Test
Exception in thread "main" java.lang.IllegalArgumentException
        at java.lang.Thread.setPriority(Unknown Source)
        at Test.main(Test.java:7)

D:\Yellaswamy_ClassNotes\UNIT-3\Top10Exceptions>
----------------------------------------
```

finally Example program

```
==========================
class FinallyExample
{
static void m1()
{
        try
        {
                System.out.println("inside m1() method");
                throw new RuntimeException("Test");
        }
        finally
        {
                System.out.println("m1 method finally block");
        }
}

static void m2()
{
        try
        {
                System.out.println("inside m2() method");
                return;
        }
        finally
        {
                System.out.println("m2 method finally block");
        }
```

```
}
static void m3()
{
	try
	{
		System.out.println("inside m3() method");
		throw new RuntimeException("Test");
	}
	finally
	{
		System.out.println("m3 method finally block");
	}
}

public static void main(String[] args)
	{
	try
	{
		m1();
	}
	catch (Exception e)
	{
		System.out.println("Exception caught");
	}
	m2();
	m3();
	}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-3>javac FinallyExample.java

D:\Yellaswamy_ClassNotes\UNIT-3>java FinallyExample
inside m1() method
m1 method finally block
Exception caught
inside m2() method
m2 method finally block
inside m3() method
```

```
m3 method finally block
Exception in thread "main" java.lang.RuntimeException: Test
        at FinallyExample.m3(FinallyExample.java:33)
        at FinallyExample.main(FinallyExample.java:52)

D:\Yellaswamy_ClassNotes\UNIT-3>
```

Customized or User Defined Exception Handling

==========

```
class TooYoungException extends RuntimeException
{

TooYoungException(String s)
{
super(s);
}
}

class TooOldException extends RuntimeException
{

TooOldException(String s)
{
super(s);
}
}

class CustExceptionDemo
{
public static void main(String[] args)
{
int age=Integer.parseInt(args[0]);
if(age>60)
        {
        throw new TooYoungException("please wait some more time you will get best match");

        }
        else if(age<18)
        {
```

```
        throw new TooOldException("Your age already crossed marriage age & no chance of getting
marriage");

        }
        else
        {
                System.out.println("You will get match details soon by email");
        }
}
}
```

throws Keyword:
==========
To delegate the responsibility to the caller.

```
import java.io.*;
class Test
{
public static void main(String args[])
{
//PrintWriter out=new PrintWriter("Hello.txt");
//out.println("Hello");
try
{
        Thread.sleep(10000);
}
catch (InterruptedException e)
{
}

}
}
```

```
/*

D:\Yellaswamy_ClassNotes\UNIT-3\throws>javac Test.java
Test.java:6: unreported exception java.io.FileNotFoundException; must be caught
or declared to be thrown
PrintWriter out=new PrintWriter("Hello.txt");
```

```
            ^
1 error

D:\Yellaswamy_ClassNotes\UNIT-3\throws>

*/

//--------------------

/*D:\Yellaswamy_ClassNotes\UNIT-3\throws>javac Test.java
Test.java:8: unreported exception java.lang.InterruptedException; must be caught
 or declared to be thrown
Thread.sleep(10000);
          ^
1 error

D:\Yellaswamy_ClassNotes\UNIT-3\throws>
*/


//1.By using try-catch
//2.By using throws keyword



import java.io.*;
class Test1
{
public static void main(String args[])throws InterruptedException
{


            Thread.sleep(10000);

}
}
-----------------------------
class Test2
{
public static void main(String[] args)throws InterruptedException
{
```

```
        doStuff();
}
public static void doStuff()throws InterruptedException
        {
        doMoreStuff();
        }
public static void doMoreStuff()throws InterruptedException
        {
        Thread.sleep(10000);
        }
}
=======================================
class Test
{
        public static void main(String[] args)
        {
                //throw new Exception();
                throw new Error();
        }
}
----------------------------------------

1.try-with Resources
====================
import java.io.*;
class TryWithResources
{
public static void main(String[] args) throws Exception
{
        try(BufferedReader br=new BufferedReader(new FileReader("input.txt")))
        {
        }


}
}



2.Multi-catch block.
====================
```

```
	import java.io.*;
class MultiCatchBlock
{

public static void main(String[] args)
	{
	try
	{
		//System.out.println(10/0);
		String s=null;
		System.out.println(s.length());
	}
	catch (ArithmeticException|NullPointerException e)
	{
		System.out.println(e);
	}
	}
}
```