What is Exception?
How to handle the Exception?
How many types of Exception?
What is difference b/w Exception and Error?
  Exception : mistakes which can be handled programmatically
  Error   : mistakes which can't be handled programatically(occurs due to lack of System Resources)
Information of Exceptionhierarcy

```
                                    Throwable(P)
      Exception(partiallychecked)                                      Error
          RunTimeException(unchecked)     IOException InterruptedException
      OutOfMemoryError
            a. NFE
            b. IOBE
            c. ArithmeticException
            d. NullPointerException
```

Need of finally block
      a. try block   -> risky code(we can't gaurantee all the statements present in try will be executed or not)
      b. catch block -> handling code(this block executes only when excpetion occurs)
                    we can't gaurantee all the statements present in catch block will be executed or not)
      c. finally block -> Compulsory executed block whetere exception occurs or doesn't occurs
                        Exception occurs -> not handled then also execute
                        Exception occurs -> handled then also execute.

Difference b/w final,finally,finalize method
      final   => applied on variable,class,methods
      finally => block meant for resource releasing logic
      finalize => meant for resource releasing logic for an object which will be called by "Garbage Collector".

throw keyword in java
====================
This keyword is used in java to throw the exception object manually and informing jvm to handle the exception.
      Syntax:: throw new ArithmeticException("/ by zero");

Eg#1.

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

Here the jvm will generate an Exception called "ArithmeticException", since main() is not handling
it will handover the control to jvm, jvm will handover to DEH to dump the exception object details
through printStackTrace().

                  vs

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

Here the programmer will generate ArithmeticException,and this exception object will be delegated
to JVM, jvm will handover the control to DEH to dump the excpetion information details through
printStackTrace().

Note:: throw keyword is mainly used to throw an customized exception not for predefined exception.

eg::
```
  class Test{
      static ArithmeticException e =new ArithmeticException();
      public static void main(String... args){
            throw e;
      }
  }
```
Output::
```
   Exception in thread "main" java.lang.ArithmeticException
```

eg::
```
class Test{
      static ArithmeticException e; // null
      public static void main(String... args){
            throw e;//NullPointerException
      }
  }
```

Output::
```
  Exception in thread "main" java.lang.NullPointerException.
```

Case2
=====
After throw statement we can't take any statement directly otherwise we will get compile time
error saying unreachable statement.

eg#1.
```
class Test{
      public static void main(String... args){
            System.out.println(10/0);
            System.out.println("hello");
      }
}
```
Output::
```
Exception in thread "main" java.lang.ArithmeticException
```
                     VS

eg#2.
```
class Test{
  public static void main(String... args){
            throw new ArithmeticExeption("/ by zero");
            System.out.println("hello");
  }
}
```
Output::
```
 CompileTime error
  Unreachable statement
    System.out.println("hello");
```

eg#1.

```
class Test3{
      public static void main(String... args){
            throw new Test3();
      }
}
Output::
 Compile time error.
 found::Test3
 required:: java.lang.Throwable
```

eg#2.

```
public class Test3 extends RunTimeException{
      public static void main(String... args){
            throw new Test3();
      }
}
Output::
  RunTimeError: Exception in thread "main" Test3
```

Customized Exceptions (User defined Exceptions)
===
Sometimes we can create our own exception to meet our programming requirements.
Such type of exceptions are called customized exceptions (user defined exceptions).
Example:
1. InSufficientFundsException
2. TooYoungException
3. TooOldException

Program:

```
class TooYoungException extends RuntimeException{
      TooYoungException(String s){
            super(s);
      }
}
class TooOldException extends RuntimeException{
      TooOldException(String s){
            super(s);
      }
}
class CustomizedExceptionDemo{
      public static void main(String[] args){

      int age=Integer.parseInt(args[0]);
      if(age>60){
          throw new TooYoungException("please wait some more time.... u will get
best match");
        }
      else if(age<18){
            throw new TooOldException("u r age already crossed....no chance of
getting married");
      }
      else{
            System.out.println("you will get match details soon by e-mail");
      }
  }
}
Output
======
1)E:\corejava>java CustomizedExceptionDemo 61
```

```
      Exception in thread "main" TooYoungException: please wait some more time....
u will get  best match at
CustomizedExceptionDemo.main(CustomizedExceptionDemo.java:21)
2)E:\corejava>java CustomizedExceptionDemo 27
      You will get match details soon by e-mail

3)E:\corejava>java CustomizedExceptionDemo 9
      Exception in thread "main" TooOldException: u r age already crossed....no
chance of  getting married at CustomizedExceptionDemo.main
                  (CustomizedExceptionDemo.java:25)
```

Note: It is highly recommended to maintain our customized exceptions as unchecked by extending RuntimeException.

throws statement
================
 In our program if there is a chance of raising checked exception then compulsory we should handle
 either by try catch or by throws keyword otherwise the code won't compile.

eg#1.
```
import java.io.*;
class Test3{
      public static void main(String... args){
            PrintWriter pw=new PrintWriter("abc.txt");
            pw.println("Hello world");
      }
}
```
CE: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown

eg#2.
```
class Test3{
      public static void main(String... args){
            Thread.sleep(3000);
      }
}
```
CE: unreported exception java.lang.InterruptedException; must be caught or declared to be thrown

We can handle this compile time error by using the following 2 ways
1. using try catch
2. using throws keyword

1. using try catch
```
class Test3{
      public static void main(String... args){
            try{
                  Thread.sleep(5000);
            }catch(InterruptedException ie){}
      }
}
```
ouput:: compiles and succesfully running

2. using throws keyword
```
class Test{
      pubilc static void main(String... args) throws InterruptedException{
            Thread.sleep(5000);
      }
}
output:: compiles and succesfully running.
=> we can use throws keyword to delegate the responsibility of exception handling
to the caller
    method. Then caller method is responsible to handle the exception.

Note::
  . Hence the main objective of "throws" keyword is to delegate the responsiblity
of exception handling to the caller method.
  . throws keyword required only for checked exception. usage of throws keyword for
unchecked exception there is no use.
  . "throws" keyword required only to convince compiler.Usage of throws keyword
does not prevent abnormal termination of the program.

Hence recomended to use try-catch over throws keyword.

eg#1.
class Test{
      public static void main(String... args) throws InteruptedException{
            doWork();
      }
      pubilc static void doWork() throws InteruptedException{
            doMoreWork();
      }
      public static void doMoreWork() throws InteruptedException{
            Thread.sleep(5000);
      }
}

In the above code, if we remove any of the throws keyword it would result in
"CompileTimeError".


                                          Throwable
                                        |

==========================================================================
            |                                                         |
Exception(can be handled)                                        Error(can't be
handled)
 a. CheckedException[handling code is required during compilation]
 b. UnCheckedException[RuntimeException]
      [handling code is not required during compilation]

throw keyword   -> not applicable for pre-defined exceptions(wheter it is checked
or unchecked),meant for userdefined exceptions
throws keyword  -> to handle checked exception by the caller inform compiler that
if exception gets generated it would handled
               by caller through "throws" keyword.


eg::
main() throws IOException
 doStuff()

doStuff() throws IOException
   doMoreStuff()

doMoreStuff() throws IOException
```

```
    new File("abc.txt");

Case studies of Throwable
=========================
Case 1::
    we can use throws keyword only for Throwable types otherwise we will get
compile time error.

class Test3{
      public static void main(String... args)throws Test3{

      }
}
output:Compile Time Error,Test3 cannot be Throwable

class Test3 extends RuntimeException{
      public static void main(String... args)throws Test3{

      }
}
output:Compiles and run succesfully


Case2::
public class Test3 {
      public static void main(String... args) {
            throw new Exception();
      }
}
Output:Compile Time Error
    unreported Exception must be caught or declared to be thrown

public class Test3 {
      public static void main(String... args) {
            throw new Error();
      }
}
Output:RunTimeException
 Exception in thread "main" java.lang.Error
        at Test3.main(Test3.java:4)

Case3::
  In our program with in try block,if there is no chance of rising an exception
then we can't
  write catch block for that exception, otherwise we will get Compile Time Error
saying
   "exception XXX is never thrown in the body of corresponding try statement", but
this rule
  is applicable only for fully checked exceptions only.

eg#1.
 public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (Exception e)
            {

            }
      }
}
Output:hiee

eg#2.
public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (ArithmeticException e)
            {

            }
      }
}
Output:hiee

eg#3.
public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (java.io.FileNotFoundException e)
            {

            }
      }
}
Ouput::Compile time error(fully checked Exception)

eg#4.
public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (InteruptedException e)
            {

            }
      }
}
Ouput::Compile time error(fully checked Exception)
        exception InterruptedException is never thrown in the body of correspoding
```

try statement.

eg#5.
```
public class Test3
{
      public static void main(String... args) {

            try
            {
                  System.out.println("hiee");
            }
            catch (Error e)
            {

            }
      }
}
Output:hiee
```

Case4:: we can use throws keyword only for constructors and methods but not for classes.

eg#1.
```
class Test throws Exception//invalid
{
      Test() throws Exception{//valid

      }
      methodOne() throws Exception{//valid

      }
}
```

Exception handling keywords summary
===================================
1. try => maintain risky code
2. catch=> maintain handling code
3. finally=> maintain cleanup code
4. throw => To hanover the created exception object to JVM manually
5. throws=> To delegate the Exception object from called method to caller method.

Various compile time errors in ExceptionHandling
==
1. Exception XXX is already caught
2. Unreported Exception XXX must be caught or declared to be thrown.
3. Exception XXX is never thrown in the body of corresponding try statement.
4. try without catch,finally
5. catch without try
6. finally without try
7. incompatible types :found xxx
                   required:Throwable
8. unreachable code.

Exception which are normally occured in java coding
===
1. Based on the events occured exceptions are classified into 2 types
    a. JVM Exceptions
    b. Programtic Exceptions

```
JVM Exceptions
  => The exceptions which are raised automatically by the jvm whenever a particular
event occurs are called JVM Exceptions
     eg:: ArrayIndexOutOfBoundsException
          NullPointerException

ProgramaticExceptions
  => The exceptions which are raised explicitly by the programmer or by API
developers is called as "Programatic Exceptions".
      eg:: IllegalArgumentException,NumberFormatException

ArrayIndexOutOfBoundsException
   =>This exception is raised automatically whenever we are trying to access array
elements which is out of the range.


Top-10-JavaExceptions
===================
1. ArithmeticException
2. NullPointerException
3. StackOverFlowError
4. IllegalArgumentException
      eg:: Thread t=new Thread();
         t.setPriority(10);
         t.setPriority(100);//invalid
5. NumberFormatException
6. ExceptionInInitializerError
7. ArrayIndexOutOfBoundsException
8. NoClassDefFoundError
9.  ClassCastException
10. IllegalStateException(learn in servlet programming)
11. AssertionError(learn in Junit)


JVM-Exception
============
 a. ArithmeticException
 b. NullPointerException
 c. ArrayIndexOutOfBoundsException
 d. StackOverFlowError
 e. ClassCastException
 f. ExceptionInInitalizerError

Programmatic-Exception
=====================
 a. IllegalArgumentException
 b. NumberFormatException
 c. IllegalStateException
 d. AssertionError

eg::
public class Test {
      static{
            String name = null;
            System.out.println(name.length());//ExceptionInInitializerError
      }
      public static void main(String[] args) {
      }
}
```

```
eg::
public class Test {
      static int i = 10/0;//ExceptionInInitializerError
      public static void main(String[] args) {
      }
}
```

1.7 version Enhancements
========================
 1. try with resource
 2. try with multicatch block

untill jdk1.6, it is compulsorily required to write finally block to close all the resources which are open as a part of try block.

```
eg:: BufferReader br=null;
      try{
       br=new BufferedReader(new FileReader("abc.txt")); //risky code
      }catch(IOException ie){
          ie.printStackTrace();//handling code
      }finally{
      //resource releasing code
       try{
         if(br!=null){
            br.close();
         }
          }catch(IOException ie){
            ie.printStackTrace();
        }
      }
```

Problems in the apporach
========================
1. Compulsorily the programmer is required to close all opened resources which increases the complexity of the program
2. Compulsorily we should write finally block explicitly, which increases the length of the code and reviews readablity.
   To Overcome this problem SUN MS introduced try with resources in "1.7" version of jdk.

try with resources
==================
In this apporach, the resources which are opened as a part of try block will be closed automatically once the control reaches
to the end of try block normally or abnormally,so it is not required to close explicitly so the complexity of the program
would be reduced.
It is not required to write finally block explicitly,so length of the code would be reduced and readability is improved.

```
try(BufferedReader br=new BufferedReader(new FileReader("abc.txt")){
      //use br and perform the necessary operation
        //once the control reaches the end of try automatically br will be closed

}catch(IOException ie){
     //handling code
}
```

```
Rules of using try with resource
================================
1. we can declare any no of resources, but all these resources should be seperated
with ;
 eg#1.
   try(R1;R2;R3;){
         //use the resources
   }

2. All resources are said to be AutoCloseable resources iff the class implements an
interface called "java.lang.AutoCloseable"
   either directly or indirectly
         eg:: java.io package classes, java.sql.package classes

3. All resource reference by default are treated as implicitly final and hence we
can't perform reassignment with in try block.
      try(BufferedReader br=new BufferedReader(new FileWriter("abc.txt")){
            br=new BufferedReader(new FileWriter("abc.txt"));
      }
    output::CE: can't reassign a value

4. untill 1.6 version try should compulsorily be followed by either catch or
finally, but from
   1.7 version we can take only take try with resources without cath or finally.
         try(R){
            //valid
         }

5. Advantage of try with resources concept is finally block will  become dummy
because we are not  required to close
   resources explicitly.


MultiCatchBlock
===============
Till jdk1.6, eventhough we have multiple exception having same handling code we
have to write a
seperate catch block for every exceptions, it increases the length of the code and
reviews
readability.
try{
   ....
   ....
   ....
   ....
}catch(ArithmeticException ae){
      ae.printStackTrace();
}catch(NullPointerExcepion ne){
      ne.printStackTrace();
}catch(ClassCastException ce){
      System.out.println(ce.getMessage());
}catch(IOException ie){
      System.out.println(ie.getMessage());
}

To overcome this problem SUNMS has introduced "Multi catch block" concept in 1.7
version
try{
   ....
   ....
   ....
   ....
}catch(ArithmeticException |NullPointerException e){
      e.printStackTrace();
}catch(ClassCastException |IOException e){
      e.printStackTrace();
}
```

In multicatch block,there should not be any relation b/w exception types(either child to parent
or parent to child or same type) it would result in compile time error.

```
eg:: try{

     }catch( ArithmeticExeption | Exception e){
           e.printStackTrace();
     }
Output:CompileTime Error
```

Exception Propogation
=====================
 With in a method, if an exception is raised and if that method does not handle that exception
 then Exception object will be propogated to the caller method then caller method is responsible
 to handle that exceptions, This process is called as "Exception Propogation".

```
eg::
JVM ===> ArithmeticException occured ====> DEH ====> e.printStackTrace()
  |
main()
 doStuff()//ArithmeticException occured

doStuff()
 doMoreStuff()//ArithmeticException occured

doMoreStuff()
     ArithmeticException occured
```

ReThrowing an Exception
=====================
To convert one exception type to another exception type, we can use rethrowing exception concept.

```
eg::
public class TestApp
{
      public static void main(String[] args)
      {
        try{
            System.out.println(10/0); //ArithmeticException
          }catch( ArithmeticException  e){
            throw new NullPointerException();//Creating a new
Exception[NullPointerException] and throwing it to the caller.
          }
      }
}
Output::
```

```
Exception in thread "main" java.lang.NullPointerException
        at TestApp.main(TestApp.java:10)
```