Exception :
****
An unwanted, unexcepted event that disturbs normal flow of program is called as exception.
for ex., now we are talking on call and if your or my network will not work then we will not
able to listen eachother voice. Because of network issue our normal flow of conversation get distrubted
this is called exception.
And if i take a technical example if you want to read a data from remote file which is in Japan but that
file is not present there then you will get exception that is FileNotFoundException.

Purpose of Exception Handling :
****
- Graceful termination of program
for ex., you have to submit the project at 9 am that's you are doing work on that program from 4 am and your
project task will complete at 8 am but at 7.45 am electricty gone then that time u will not able to save your
project in your system and after coming electicity your project will not be there. so avoid this problem you have
to take backup allready then you can save your program.
If i go with technical example if in your program there is DB connection which is open connection, read data and close data
but at the time of we get SQL Exception then our program stopped abnormally to avoid this problem we have to handle exception.

Meaning of Exception Handling :
****
Defining alternative way to handle exception is called as exception handling
for ex., i have to travel and which bus ticket i booked that bus is not available then i can find the alternative way
to go from other bus or train. this is called exception handling.
And if i take a technical example if you want to read a data from remote file which is in Japan but that
file is not present there then you will get exception that is FileNotFoundException. but you allready handled that exception
which is if Japan file is not there then take file from your local system.

Runtime Stack Mechanisam :
****
- For every thread, JVM (Java virtual machine) creates a run-time stack.
1) Each and every call performed in a thread is stored in the stack.
2) Each entry in the run-time stack is known as an activation record or stack frame.
3) After completing every method call by the thread is removed from the corresponding entry of the stack.
4) After completing all the methods, the stack will be empty and that run-time stack will be destroyed by the JVM before

terminating the thread.

- Letâ€™s have a look at the below program to understand the working of the run-time stack

- Construction of run-time Stack :
1) Firstly, the main thread will call the main() method, and the corresponding entry will be in the stack.
2) After that main() method is called the fun() method, which will store in the stack.
3) In the fun() method, moreFun() method is called. Therefore at last moreFun() will be stored in the stack.
4) Finally, moreFun() is not calling any method and it will print Hello Geeks!

```
class Geeks {
    public static void main(String[] args)
    {
        fun();
    }

    public static void fun()
    {
        moreFun();
    }

    public static void moreFun()
    {
        System.out.println("Hello Geeks!");
    }
}
```

Destruction of the run-time stack:
After printing Hello Geeks!, its corresponding entry will be removed from the stack and it will go to the fun() method and
there is nothing for execution thatâ€™s why the entry of fun() method is removed from the stack and so on. When the stack
is empty then the run-time stack is destroyed by the JVM.

Default Exception Handling :
****
If we didn't handled exception then JVM will call default exception handler and we will get which exception occurred on which method and which line.
ex.,

```
public class ExceptionDemo {
    public static void doStuff() {
        domoreStuff();
    }
    public static void domoreStuff() {
        System.out.println(10/0);
    }

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

```
}

//op
/*
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at ExceptionDemo.domoreStuff(ExceptionDemo.java:6)
      at ExceptionDemo.doStuff(ExceptionDemo.java:3)
      at ExceptionDemo.main(ExceptionDemo.java:10)
 */


Exception Hierarchy :
**********************
Throwable (Class) it's a root exception
1) Exception
    i) RuntimeException
        a) AE
        b) NPE
        c) CCE
        d) IndexOutOfBoundException
    ii) IOException
    iii) InterruptedException

2) Error
    i) VM Error
        a) StackOutOfError
        b)

Difference between Exception vs Error :
****************************************
Exception :
- exception occured because of our program only
- we can handle these exception
- exceptions are recoverable

Error :
- error not caused by our program
- we can't handle the error
- errors are not recoverable
- it's occure because of lack of resources

Q. Difference between Checked and Unchecked Exception.
******************************************************

There is a concept that the compile time exceptions occur at the compile
time and runtime exceptions occur
at runtime. This is a wrong concept. The right concept is that all the
exceptions occur at the runtime only.

Checked Exception(Compile Time Exception) :
    1. Checked Exceptions are the exceptions that are checked and handled
at compile time.
    2. The program gives a compilation error if a method throws a checked
exception.
```

3. If some code within a method throws a cheked exception, then the method must either handle
the exception or it must specify the exception using throws keyword.
4. A checked exceptions occur when the chances of failure are too high.
5. They are direct subclass of Exception class but do not inherit from RuntimeException.
6. Ex.,

```
import java.io.FileInputStream;

public class Demo {
	public static void main(String[] args) {
		FileInputStream fis = new FileInputStream("abc.txt");
	}
}
```

above example will throw error if abc.txt file is not present. For not getting error at the runtime we have to
handle that error. For the we have to use try__catch block.
Solution of above program is :

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class Demo {
	public static void main(String[] args) {
		try {
			FileInputStream fis = new FileInputStream("abc.txt");
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}
```

Unchecked Exception(RunTime Exception) :
1. Unchecked Exceptions are the exceptions that are not checked and handled at compile time.
2. The program compiles fine because the compiler is not able to check the exception.
3. A method is not forced by compiler to declare the unchecked exceptions thrown by its
implementation. Generally, such methods almost always do not declare them, as well.
4. Unchecked exceptions occurs mostly due to programming mistakes.
5. They are direct subclass of RuntimeException class.
6. Example.,

```
public class RuntimeException {

	public static void main(String[] args) {
		int a = 100, b = 0, c;
		c = a/b;
		System.out.println(c);
	}
}
```

//In this program compile cab't check the exception and can compile the program successfully and
// will show an Arithmetic Exception.

Real World Example : In real world your mother asks for your purse or your company id-card so that
you won't face any problem due to these things means at compile time before you went to office your
mother is checking the things (considered as exceptions) due to which at runtime(which is your company
area) you can get any problem or exception.

Fully checked exception vs Partially checked exception :
****
Fully checked :
ex., you are going to airport with your kids for travelling Mumbai to USA then there security guard will check u also and your kids also
this type of exception called fully checked
ex., IOException, InterruptedException, RuntimeException (except Throwable and Exception all are fully checked)

Partially checked :
Ex., you are going to mall with your kids then there security guard will check only you not your kids this type of checking is called partially checked
ex., Throwable, Exception (only these two are partially checked)

Describe the behaviour of the following exception :
****

1. IOException : checked (Fully checked)
2. RuntimeException : unchecked
3. InterruptedException : checked(Fully checked)
4. Error : unchecked
5. Throwable : unchecked (partially checked)
6. ArithmeticException : unchecked
7. NullPointerException : unchecked
8. Exception : checked (Partially checked)
9. FileNotFoundException : checked (Fully checked)

Control Flow of try catch block :
****
Ex.,

```
        try {
            System.out.println("Statement-1");
            System.out.println("Statement-2");
            System.out.println("Statement-3");
        }
        catch (Exception e) {
            System.out.println("Statement-4");
        }
        System.out.println("Statement-5");
```

Case-1 : If there is no exception (1, 2, 3, 5) Normal termination
Case-2 : If an exception raised at statement-2 and corressponding catch block matched
(1, 4, 5) Normal termination
Case-3 : If an exception raised at statement-2 and corressponding catch block not matched
(1) abnormal termination
Case-4 : If an exception raised in statement-4 (catch block) or statement-5 abnormal termination.

Purpose and Speciality of finally block :
****
- to write the cleanup code finally block is used
ex., to close DB connection
- it will excute always if exception occured or not no matter

```
try { (Risky code)
    //open DB connection
    // read data
}
catch (Exception e) {   (Handling code)
    //exception handling
}
finally {   (Cleanup code)
    //close DB connection
}
```

Case-1 : If there is no exception finally will execute with try block
ex.,

```
try {
    System.out.println("try block");
}
catch (ArithmeticException ae) {
    System.out.println(ae);
}
finally {
    System.out.println("finally");
}
/*
try block
finally
*/
```

Case-2 : if there is exception occured finally block will execute with catch block
ex.,

```
try {
    System.out.println(10/0);
}
catch (ArithmeticException ae) {
    System.out.println(ae);
}
finally {
    System.out.println("finally");
}
```

```
        /*
        java.lang.ArithmeticException: / by zero
        finally
        */
```

Case-3 : If exception occurred and you didn't handled that exception then finally block will execute but program terminate abnormally
ex.,

```
        try {
            System.out.println(10/0);
        }
        finally {
            System.out.println("finally");
        }
        /*
        finally
        Exception in thread "main" java.lang.ArithmeticException: / by
zero
           at ExceptionDemo.main(ExceptionDemo.java:4)
        */
```

Methods to print Exception :
****
1) e.printStackTrace()
2) e.toString(); print(e); print(e.toString())
3) print(e.getMessage())

Ex.,

```
public static void main(String[] args) {
        try {
            System.out.println(10/0);
        }
        catch (ArithmeticException e) {
            //System.out.println(e);      //if you want name of exception
and description then go for this
            /*
            java.lang.ArithmeticException: / by zero
            End statement
             */

            //System.out.println(e.toString());     //name of exception
and description
            /*
            java.lang.ArithmeticException: / by zero
            End statement
             */

           //e.printStackTrace();       //complete information of
exception like which, what and on which line
            /*
            End statement
            java.lang.ArithmeticException: / by zero
              at ExceptionDemo.main(ExceptionDemo.java:4)
             */
```

```
            System.out.println(e.getMessage());     //to print only
description
            /*
            / by zero
            End statement
             */
        }
        System.out.println("End statement");
}
```

try with multiple catch block :
****
1) This is a wrong way because here is chances to occure Arithmetic exception but
first you are taking Parent Exception and then ArithmeticException

```
       try {
            System.out.println(10/0);
        }
        catch (Exception e) {
            System.out.println(e);
        }
        catch (ArithmeticException ae) {
            System.out.println(ae);
        }

        //op : java: exception java.lang.ArithmeticException has already
been caught
```

2) Correct way :
ex.,

```
        try {
            System.out.println(10/0);
        }
        catch (ArithmeticException ae) {
            System.out.println(ae);
        }
        catch (Exception e) {
            System.out.println(e);
        }
```

Finally vs return statement :
****
- finally block dominates return statement
- finally block will always run if there is return statement or not in try or catch block.
- if there is return statement in try block then it will print before return statement code then print finally block output and then again it will go on try block return statement
ex.,

```
public static void main(String[] args) {
        try {
            System.out.println("try block");
```

```
            return;
        }
        catch (Exception e) {
            System.out.println(e);
            return;
        }
        finally {
            System.out.println("finally block");
        }

        /*
        try block
        finally block
         */
}
```
- if there is return statement present in all three blocks then finally block return statement get priority

ex.,

```
public static String priority() {
        try {
            return "try block";
        }
        catch (Exception e) {
            return "catch block";
        }
        finally {
            return "finally block";
        }
}
public static void main(String[] args) {
     System.out.println(priority());
}
// op : finally block
```

Finally vs system.exit() :

****

- System.exit(0) dominate finally block (JVM going to shut down that's why program will stop)
- here 0 is a status code
- 0 means normal termination and non-zero means abnormal termination
- there is only one situation when finally block will not execute that is system.exit(0)

ex.,

```
public static void main(String[] args) {
        try {
            System.out.println("try block");
            System.exit(0);
        }
        catch (Exception e) {
            System.out.println(e);
        }
        finally {
            System.out.println("finally block");
```

```
        }
}
// op : try block

control flow in try-catch-finally :
***********************************
Case-1 : If there is no exception try and finally block will run
ex.,
    public static void main(String[] args) {
        try {
            System.out.println("statement - 1");
            System.out.println("statement - 2");
            System.out.println("statement - 3");
        }
        catch (Exception e) {
            System.out.println("statement - 4");
        }
        finally {
            System.out.println("statement - 5");
        }
        System.out.println("statement - 6");
        /*
        statement - 1
        statement - 2
        statement - 3
        statement - 5
        statement - 6
         */
    }

Case-2 : If an exception raised at statement-2 and corresponding catch
block matched program will terminate normally
ex.,
public static void main(String[] args) {
        try {
            System.out.println("statement - 1");
            System.out.println("statement - 2" + 10/0);
            System.out.println("statement - 3");
        }
        catch (ArithmeticException ae) {
            System.out.println("statement - 4");
        }
        finally {
            System.out.println("statement - 5");
        }
        System.out.println("statement - 6");
        /*
        statement - 1
        statement - 4
        statement - 5
        statement - 6
         */
    }
```

Case-3 : If an exception raised at statement-2 and corresponding catch block not matched program will terminate abnormally
ex.,

```
public static void main(String[] args) {
        try {
            System.out.println("statement - 1");
            System.out.println("statement - 2" + 10/0);
            System.out.println("statement - 3");
        }
        catch (NullPointerException ae) {
            System.out.println("statement - 4");
        }
        finally {
            System.out.println("statement - 5");
        }
        System.out.println("statement - 6");
        /*
        statement - 1
        statement - 5
        Exception in thread "main" java.lang.ArithmeticException: / by
zero
              at ExceptionDemo.main(ExceptionDemo.java:5)

         */
    }
```

Case-4 : If an exception raised at statement-4

Case-5 : if an exception raised at statement-5 or statement-6 then it will abnormal termination
ex.,

```
public static void main(String[] args) {
        try {
            System.out.println("statement - 1");
            System.out.println("statement - 2");
            System.out.println("statement - 3");
        }
        catch (Exception e) {
            System.out.println("statement - 4" );
        }
        finally {
            System.out.println("statement - 5" + 10/0);
        }
        System.out.println("statement - 6");
        /*
        statement - 1
        statement - 2
        statement - 3
        Exception in thread "main" java.lang.ArithmeticException: / by
zero
              at ExceptionDemo.main(ExceptionDemo.java:12)
         */
    }
```

Nested try-catch-finally :
****
- if we have to many risky statement in try block then use nested try-catch for that code
- Advantage : if exception is raising for specific statement then that code catch block will execute
and after that remaining statement will execute. if we didn't did like this then execution stop on that statement only.
- ex.,

```
public static void main(String[] args) {
        try {
            System.out.println("Outer try");
            try {
                System.out.println("Inner try");
                System.out.println(10/0);
            }
            catch (ArithmeticException ae) {
                //if in this block exception is not related to try block then outer catch block will execute
                System.out.println("Inner exception handled");
            }
        }
        catch (Exception e) {
            System.out.println("Outer exception handled");
        }
        finally {
            System.out.println("Finally block");
        }

        /*
        Outer try
        Inner try
        Inner exception handled
        Finally block
         */
}
```

- ex.,

```
    public static void main(String[] args) {
            try {
                System.out.println("Outer try");
                try {
                    System.out.println("Inner try");
                    System.out.println(10/0);
                }
                catch (NullPointerException ae) {
                    //if in this block exception is not related to try block then outer catch block will execute
                    System.out.println("Inner exception handled");
                }
            }
            catch (Exception e) {
                System.out.println("Outer exception handled");
            }
```

```
            finally {
                System.out.println("Finally block");
            }

            /*
            Outer try
            Inner try
            Outer exception handled
            Finally block

             */
    }
```

Case-1 : If there is no exception : try block and finally block will execute
ex.,

```
    public static void main(String[] args) {
            try {
                System.out.println("Outer try");
                try {
                    System.out.println("Inner try");
                }
                catch (NullPointerException ae) {
                    System.out.println("Inner exception handled");
                }
                System.out.println("after inner try block statement");
            }
            catch (Exception e) {
                System.out.println("Outer exception handled");
            }
            finally {
                System.out.println("Finally block");
            }

            /*
            Outer try
            Inner try
            after inner try block statement
            Finally block
             */
    }
```

Case-2 : If an exception raised at statement-2 and corressponding catch block matched
- ex.,

```
    public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2 : "+10/0);
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5");
```

```
                    System.out.println("Statement-6");
                }
                catch (Exception e) {
                    System.out.println("Statement-7");
                }
                finally {
                    System.out.println("Statement-8");
                }
                System.out.println("Statement-9");
            }
            catch (Exception e) {
                System.out.println("Statement-10");
            }
            finally {
                System.out.println("Statement-11");
            }
            System.out.println("Statement-12");
    }
        /*
        Statement-1
        Statement-10
        Statement-11
        Statement-12
        Normal termination
         */
```

Case-3 : If an exception raised at statement-2 and corresspoinding catch block not matched.

- ex.,

```
    public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2 : "+10/0);
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5");
                    System.out.println("Statement-6");
                }
                catch (Exception e) {
                    System.out.println("Statement-7");
                }
                finally {
                    System.out.println("Statement-8");
                }
                System.out.println("Statement-9");
            }
            catch (NullPointerException e) {    //not matching catch
block
                System.out.println("Statement-10");
            }
            finally {
                System.out.println("Statement-11 : outer finally block");
```

```
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
    }
        /*
    Statement-1
    Statement-11 : outer finally block
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at ExceptionDemo.main(ExceptionDemo.java:5)

         */

Case-4 : If an exception raised at statement-5 and corressponding ineer
catch block matched
- ex.,
public static void main(String[] args) {
        try {
            System.out.println("Statement-1");
            System.out.println("Statement-2");
            System.out.println("Statement-3");

            try {
                System.out.println("Statement-4");
                System.out.println("Statement-5 : "+10/0);
                System.out.println("Statement-6");
            }
            catch (Exception e) {
                System.out.println("Statement-7 : inner catch block");
            }
            finally {
                System.out.println("Statement-8");
            }
            System.out.println("Statement-9");
        }
        catch (Exception e) {
            System.out.println("Statement-10 : outer catch block");
        }
        finally {
            System.out.println("Statement-11 : outer finally block");
        }
        System.out.println("Statement-12");
        System.out.println("Normal termination");
}
    /*
Statement-1
Statement-2
Statement-3
Statement-4
Statement-7 : inner catch block
Statement-8
Statement-9
Statement-11 : outer finally block
Statement-12
Normal termination
```

```
     */


Case-5 : If an exception raised at statement-5 and inner catch block not
matched but outer catch block matched
- ex.,
    public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2");
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5 : "+10/0);
                    System.out.println("Statement-6");
                }
                catch (NullPointerException e) {        //not matching
catch block
                    System.out.println("Statement-7 : inner catch
block");
                }
                finally {
                    System.out.println("Statement-8 : Inner finally
block");
                }
                System.out.println("Statement-9");
            }
            catch (Exception e) {
                System.out.println("Statement-10 : outer catch block");
            }
            finally {
                System.out.println("Statement-11 : outer finally block");
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
    }
        /*
    Statement-1
    Statement-2
    Statement-3
    Statement-4
    Statement-8 : Inner finally block
    Statement-10 : outer catch block
    Statement-11 : outer finally block
    Statement-12
    Normal termination
         */

Case-6 : If an exception raised at statement-5 and both inner catch and
outer catch block not matched
- ex.,
    public static void main(String[] args) {
            try {
```

```
                System.out.println("Statement-1");
                System.out.println("Statement-2");
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5 : "+10/0);
                    System.out.println("Statement-6");
                }
                catch (NullPointerException e) {        //not matching
catch block
                    System.out.println("Statement-7 : inner catch
block");
                }
                finally {
                    System.out.println("Statement-8 : Inner finally
block");
                }
                System.out.println("Statement-9");
            }
            catch (NullPointerException e) {        //not matching
                System.out.println("Statement-10 : outer catch block");
            }
            finally {
                System.out.println("Statement-11 : outer finally block");
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
    }
        /*
    Statement-1
    Statement-2
    Statement-3
    Statement-4
    Statement-8 : Inner finally block
    Statement-11 : outer finally block
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at ExceptionDemo.main(ExceptionDemo.java:10)
         */

Case-7 : If an exception raised at statement-7 and corressponding catch
block matched
- ex.,
    public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2");
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5 : "+10/0);
                    System.out.println("Statement-6");
                }
```

```
                catch (NullPointerException e) {
                    System.out.println("Statement-7 : inner catch block"
+ 10/0);
                }
                finally {
                    System.out.println("Statement-8 : Inner finally
block");
                }
                System.out.println("Statement-9");
            }
            catch (Exception e) {
                System.out.println("Statement-10 : outer catch block");
            }
            finally {
                System.out.println("Statement-11 : outer finally block");
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
    }
        /*
    Statement-1
    Statement-2
    Statement-3
    Statement-4
    Statement-8 : Inner finally block
    Statement-10 : outer catch block
    Statement-11 : outer finally block
    Statement-12
    Normal termination
         */

Case-8 : If an exception raised at statement-7 and corressoponding catch
block not matched
- ex.,
    public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2");
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5 : "+10/0);
                    System.out.println("Statement-6");
                }
                catch (NullPointerException e) {    //exception raised
                    System.out.println("Statement-7 : inner catch block"
+ 10/0);
                }
                finally {
                    System.out.println("Statement-8 : Inner finally
block");
                }
                System.out.println("Statement-9");
```

```
            }
            catch (NullPointerException e) {       //not matching block
                System.out.println("Statement-10 : outer catch block");
            }
            finally {
                System.out.println("Statement-11 : outer finally block");
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
    }
        /*
    Statement-1
    Statement-2
    Statement-3
    Statement-4
    Statement-8 : Inner finally block
    Statement-11 : outer finally block
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at ExceptionDemo.main(ExceptionDemo.java:10)
         */


Case-9 : If an exception raised at statement-8 and corressponding catch
block matched
- ex.,
    public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2");
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5");
                    System.out.println("Statement-6");
                }
                catch (Exception e) {    //exception raised
                    System.out.println("Statement-7 : inner catch
block");
                }
                finally {
                    System.out.println("Statement-8 : Inner finally
block" + 10/0);
                }
                System.out.println("Statement-9");
            }
            catch (Exception e) {
                System.out.println("Statement-10 : outer catch block");
            }
            finally {
                System.out.println("Statement-11 : outer finally block");
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
```

```
    }
        /*
    Statement-1
    Statement-2
    Statement-3
    Statement-4
    Statement-5
    Statement-6
    Statement-10 : outer catch block
    Statement-11 : outer finally block
    Statement-12
    Normal termination
         */

Case-10 : If an exception raised at statement-8 and corrossponding catch
block not matched
- ex.,
    public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2");
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5");
                    System.out.println("Statement-6");
                }
                catch (Exception e) {    //exception raised
                    System.out.println("Statement-7 : inner catch
block");
                }
                finally {
                    System.out.println("Statement-8 : Inner finally
block" + 10/0);
                }
                System.out.println("Statement-9");
            }
            catch (NullPointerException e) {    //not matching block
                System.out.println("Statement-10 : outer catch block");
            }
            finally {
                System.out.println("Statement-11 : outer finally block");
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
    }
        /*
    Statement-1
    Statement-2
    Statement-3
    Statement-4
    Statement-5
    Statement-6
```

```
    Statement-11 : outer finally block
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at ExceptionDemo.main(ExceptionDemo.java:17)

         */

Case-11 : If an exception raised at statement-9 and corressponding catch
block matched
- ex.,
    public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2");
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5");
                    System.out.println("Statement-6");
                }
                catch (Exception e) {
                    System.out.println("Statement-7 : inner catch
block");
                }
                finally {
                    System.out.println("Statement-8 : Inner finally
block");
                }
                System.out.println("Statement-9 : after inner try catch
block" + 10/0);
            }
            catch (Exception e) {
                System.out.println("Statement-10 : outer catch block");
            }
            finally {
                System.out.println("Statement-11 : outer finally block");
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
    }
        /*
    Statement-1
    Statement-2
    Statement-3
    Statement-4
    Statement-5
    Statement-6
    Statement-8 : Inner finally block
    Statement-10 : outer catch block
    Statement-11 : outer finally block
    Statement-12
    Normal termination
         */
```

Case-12 : If an exception raised at statement-9 and corressponding catch block not matched
- ex.,

```
     public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2");
                System.out.println("Statement-3");

                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5");
                    System.out.println("Statement-6");
                }
                catch (Exception e) {
                    System.out.println("Statement-7 : inner catch
block");
                }
                finally {
                    System.out.println("Statement-8 : Inner finally
block");
                }
                System.out.println("Statement-9 : after inner try catch
block" + 10/0);
            }
            catch (NullPointerException e) {        //not matching block
                System.out.println("Statement-10 : outer catch block");
            }
            finally {
                System.out.println("Statement-11 : outer finally block");
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
     }
        /*
    Statement-1
    Statement-2
    Statement-3
    Statement-4
    Statement-5
    Statement-6
    Statement-8 : Inner finally block
    Statement-11 : outer finally block
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at ExceptionDemo.main(ExceptionDemo.java:19)
         */
```

Case-14 : If an exception raised at statement-11 or statement-12
- ex.,

```
    public static void main(String[] args) {
            try {
                System.out.println("Statement-1");
                System.out.println("Statement-2");
                System.out.println("Statement-3");
```

```
                try {
                    System.out.println("Statement-4");
                    System.out.println("Statement-5");
                    System.out.println("Statement-6");
                }
                catch (Exception e) {
                    System.out.println("Statement-7 : inner catch
block");
                }
                finally {
                    System.out.println("Statement-8 : Inner finally
block");
                }
                System.out.println("Statement-9 : after inner try catch
block");
            }
            catch (Exception e) {
                System.out.println("Statement-10 : outer catch block");
            }
            finally {
                System.out.println("Statement-11 : outer finally block" +
10/0);
            }
            System.out.println("Statement-12");
            System.out.println("Normal termination");
    }
        /*
    Statement-1
    Statement-2
    Statement-3
    Statement-4
    Statement-5
    Statement-6
    Statement-8 : Inner finally block
    Statement-9 : after inner try catch block
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at ExceptionDemo.main(ExceptionDemo.java:25)
         */


Various possible combinations of try-catch-finally :
*************************************************
1) try with one catch block
try {

}
catch(Exception e) {

}

2) try with multiple catch block with different exception
try {
```

```
}
catch(Exception e) {

}
catch(Exception2 e) {

}
```

3) try with multiple catch block with same exception not possible

```
try {

}
catch(Exception e) {

}
catch(Exception e) {

}
```

4) try-catch-finally

```
try {

}
catch(Exception e) {

}
finally {

}
```

4) try-finally (ex., db connection) : some it's required

```
try {

}
finally {

}
```

5) you can't take try block without catch or finally

```
try {

}
```

5) you can't take try block without catch or finally

```
catch(Exception e) {

}
```

5) you can't take finally block without try

```
finally {

}
```

4) try-finally-catch : you will get error here because you are not following order

```
try {

}
finally {

}
catch(Exception e) {

}
```

4) You will get error in the below combination also

```
try {

}
try {

}
catch(Exception e) {

}
finally {

}
```

3) try with multiple catch block

```
try {

}
catch(Exception e) {

}
catch(Exception2 e) {

}
finally {

}
```

3) try with multiple finally block not possible

```
try {

}
catch(Exception e) {

}
finally {

}
finally {

}
```

```
3) not valid
try {

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

}

3) not valid
try {

}
catch(Exception e) {

}
System.out.println("Statement");
catch(Exception2 e) {

}

3) valid
try {
    try {

    }
    catch(Exception e) {

    }
}
catch(Exception2 e) {

}

3) valid : for the same try you can't take same exception catch block but
for different we can take
try {
    try {

    }
    catch(Exception e) {

    }
}
catch(Exception e) {

}

3) not valid because for inner try there is no catch block or finally
block
try {
    try {

    }
```

```
}
catch(Exception2 e) {

}
```

4) valid

```
try {

}
catch(Exception e) {
    try {

        }
        catch(Exception e) {

        }
}
```

4) valid

```
try {

}
catch(Exception e) {

}
finally {
    try {

    }
    catch(Exception e) {

    }
    finally {

    }
}
```

5) not valid : without taking curly braces for try, you will get error

```
try
    System.out.println("Hello");
catch(Exception e) {

}
```

6) not valid : without taking curly braces for catch, you will get error

```
try {

}
catch(Exception e)
    System.out.println("Hello");
```

7) not valid : without taking curly braces for finally, you will get error

```
try {

}
catch(Exception e) {

}
finally
    System.out.println("Hello");
```

Need of throw keyword :
****
- for customized exception. for ex., at the run time we get insufficient balance in you account you want to handle that exception
then you can use here throw keyword
- To handover our created object to JVM manually for that purpose we can use throw keyword.

without throw keyword :
==========
```
public class ExceptionDemo {
    public static void main(String[] args) {
        System.out.println(10/0);
        // in this program everything happen internally
    }
}
```
In above program we will get exception which is Arithmetic exception.

with throw keyword :
==========
If we want to do this program manually we can do like below
```
public class ExceptionDemo {
    public static void main(String[] args) {
       throw new ArithmeticException("/by zero exception with throw keyword");
        //creation of exception object explicitely and handlover to the JVM manually
        //in this program everithing happen explicitely
    }
}
```

Examples :
==========
Case-1 :
```
public class ExceptionDemo {
    static ArithmeticException e;

    public static void main(String[] args) {
       throw e;
    }

    /*
```

```
    Exception in thread "main" java.lang.NullPointerException: Cannot
throw exception because "ExceptionDemo.e" is null
      at ExceptionDemo.main(ExceptionDemo.java:7)
     */
    //if e refers null then we will get null pointer exception not
arithmetic exception
}

public class ExceptionDemo {
    static ArithmeticException e = new ArithmeticException();

    public static void main(String[] args) {
       throw e;
    }

    /*
    Exception in thread "main" java.lang.ArithmeticException
      at ExceptionDemo.<clinit>(ExceptionDemo.java:4)
     */
    //here we will get arithmetic exception because we are refer new
arithmetic exception object
}


Case-2 :
public class ExceptionDemo {
    public static void main(String[] args) {
        System.out.println(10/0);
        System.out.println("Hello");
    }
    /*
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at ExceptionDemo.main(ExceptionDemo.java:5)
     */

     //here we will get runtime exception
}

public class ExceptionDemo {
    public static void main(String[] args) {
        throw new ArithmeticException("/by zero");
        System.out.println("Hello");
    }
    /*
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at ExceptionDemo.main(ExceptionDemo.java:5)
     */

    //here we will get compile time exception because we are using throw
keyword means we are handling exception explicitly
    //compile get uncreachable statement
}
```

Case-3 :

```
public class ExceptionDemo {
    public static void main(String[] args) {
        throw new ExceptionDemo();
        /*
        java: incompatible types: ExceptionDemo cannot be converted to
java.lang.Throwable
         */
        //throw keyword is only applicable for throwable object not for
normal java class
        //here we will get compile time error
    }
}

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

    /*
    Exception in thread "main" ExceptionDemo
      at ExceptionDemo.main(ExceptionDemo.java:5)
     */
}
```

Need and Usage of throws Keyword :

****

- If in your program if any chance occur checked exception then we have to handle that exception

ex.,

```
1) public class ExceptionDemo {
    public static void main(String[] args) {
        PrintWriter pw = new PrintWriter("abc.txt");
        pw.println("Hello");
    }
    /*
    java: unreported exception java.io.FileNotFoundException; must be
caught or declared to be thrown
     */
}

2) public class ExceptionDemo {
       public static void main(String[] args) {
           Thread.sleep(1000);
       }
       /*
       java: unreported exception java.lang.InterruptedException; must be
caught or declared to be thrown
        */
   }
```

- we can handle above exception using two ways i.e. using try..catch and throws keyword

1) using throws keyword (but this is not recommended) sometime our program can terminate abnormally

```
public class ExceptionDemo {
    public static void main(String[] args) throws FileNotFoundException {
        PrintWriter pw = new PrintWriter("abc.txt");
        pw.println("Hello");
    }
}
```

2) Using try..catch

```
public class ExceptionDemo {
    public static void main(String[] args) {
        try {
            Thread.sleep(1000);
        }
        catch (InterruptedException e) {
            System.out.println("Exception handled");
        }
    }
}
```

- we can use to deligate responsibility of exception handling to the catter (JVM or another method)
- It is required only for checked exceptions and for unchecked exception there is no use
- It is required only to convince compiler and its usage does not prevent abnormal termination of the program

throws keyword across multiple methods :

****

ex., caller method is responsible to handle exception

```
public class ExceptionDemo {
    public static void main(String[] args)  {
        doStuff();
    }
    public static void doStuff() {
        doMoreStuff();
    }
    public static void doMoreStuff() throws InterruptedException{
        Thread.sleep(1000);
    }

    //in this program instead of getting error in doMoreStuff method it will get in doStuff method
    // and program will terminate abnormally
}
```

```
public class ExceptionDemo {
    public static void main(String[] args) throws InterruptedException {
        doStuff();
    }
    public static void doStuff() {
        doMoreStuff();
```

```
    }
    public static void doMoreStuff(){
        Thread.sleep(1000);
    }

    // compile time error will get in doMoreStuff method if we handled
exception in main method
}


Imp cases related to throws keyword :
************************************
Case-1 : throws keyword is only used for method not for class. If we used
we will get "{ expected error"
public class ExceptionDemo throws ArithmeticException{
    public static void main(String[] args)  {
        System.out.println(10/0);
    }
    /*
    java: '{' expected
     */
}


Case-2 : We can use throw keyword only for throwable type.
ex.,
public class ExceptionDemo {
    public static void main(String[] args) throws ExceptionDemo  {

    }
    /*
    java: incompatible types: ExceptionDemo cannot be converted to
java.lang.Throwable
     */
}
If we want to use throws for class then we have to extend Throwable or
Exception or RuntimeException to make class throwable
public class ExceptionDemo extends Throwable{
    public static void main(String[] args) throws ExceptionDemo  {

    }
}
In this example code compile fine.

Case-3 :
public class ExceptionDemo{
    public static void main(String[] args) throws ExceptionDemo  {
        throw new Exception();      //this is checked exception
        //we have to handle checked exception using only throws keyword
or try...catch block
    }
    /*
    exception java.lang.Exception; must be caught or declared to be
thrown
```

```
     */
}

public class ExceptionDemo{
    public static void main(String[] args) {
        throw new Error();   //this is unchecked exception
        //in this example we will get the runtime exception so we can
handle this using throw keyword
    }
    /*
    Exception in thread "main" java.lang.Error
      at ExceptionDemo.main(ExceptionDemo.java:7)
     */
}
```

Case-4 : If there is no chance for raising exception then there is no use to use catch block.
for other exception it will not give any error but for fully checked exception we will get compile time error

```
public static void main(String[] args) {
        //1 : here code compile fine
        try {
            System.out.println("Hello");
        }
        catch (ArithmeticException ae) {    //unchecked

        }

        //2 : here code compile fine
        try {
            System.out.println("Hello");
        }
        catch (Exception ae) {  //partially checked

        }

        //3 : here get compile time
        try {
            System.out.println("Hello");
        }
        catch (IOException ae) {        //fully checked

        }
        /*
        java: exception java.io.IOException is never thrown in body of
corresponding try statement
        */

        //4 : here get compile time error
        try {
            System.out.println("Hello");
        }
        catch (InterruptedException ae) {       //fully checked
```

```
        }
        /*
        java: exception java.lang.InterruptedException is never thrown in
body of corresponding try statement
        */

        //5 : here code compile fine
        try {
            System.out.println("Hello");
        }
        catch (Error ae) {      //unchecked

        }
}
```

Exception handling keywords summary :
****
1) try : to write or maintain risky code
2) catch : to maintain handling code
3) finally : to maintain clean up code
4) throw : to hand-over our created exception object to the JVM manually (customized exception)
5) throws : to deligate responsibility of exception handling to the caller

various possible compile time errors :
****
1) try without catch or finally
2) catch without try
3) finally without try
4) unreported exception ...; must be caught or declared to be thrown
5) exception ... is never thrown in body of corresponding try statement (for fully checked exception)
6) exception ... has already been caught (in catch bolck chain should be child to parent exception)
7) unreachable statement
8) throw, throws keyword only applicable for throwable : incompatible types : Test cannot be converted to throwable

Difference between final, finalize and finally?
****
Final :
- It is a modifier applicable for classes, variable and methods. If a class declared as a final
then we can't extend that class. i.e. we can't create child class for that class
- If a method declared as final then we can't override that method in the child class.
- If a variable declared as final then it will become constant and we can't perform re-assignment for that variable.

Finalize() :

- It is a method which is always invoked by garbage collector just before destroying an object to perform cleanup activities.

Finally :
- Finally is a block that is always associated with try catch block to maintain cleanup code

Note : Finally meant for cleanup activities related to try catch block. where a finalize() meant for cleanup activities related to object.

User defined or Customized Exception :
****
- the exception which are defined by programmer that exception called customized exception

How to define and use User Defined or Customized Exception :
****
- highly recommended to go for runtime exception

```
public class ExceptionDemo extends RuntimeException{
    ExceptionDemo(String msg) {
        super(msg);
    }

    //how to use customized exception
    public static void main(String[] args) {
        int age = 45;
        if(age>60) throw new ExceptionDemo("To young exceptionðŸŠðŸŠ");
        else if(age<18) throw new ExceptionDemo("To old exceptionðŸŠ");
        else System.out.println("Chance for registration");
    }
}
```

Top- 10 Exception :
****
1) ArrayIndexOutOfBoundException :
==================================
- it is unchecked exception. The flow is like below :
- Throwable -> RuntimeException -> IndexOutOfBoundException -> ArrayIndexOutOfBoundException
- when we are try to access element array element out of bound that time we will get this exception
- ex.,

```
public class ExceptionDemo {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 5};
        System.out.println(arr[1]);
        System.out.println(arr[100]);
    }
    /*
2
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Index 100 out of bounds for length 4
      at ExceptionDemo.main(ExceptionDemo.java:10)
     */
}

2) NullPointerException :
=========================
- when we are trying to perform any opertaion on null value then we will
get NPE.
- NullPointerException <- RuntimeException <- Exception <- Throwable
- ex.,
public class ExceptionDemo {
    public static void main(String[] args) {
        String str = null;
        System.out.println(str.length());
    }
    /*
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"String.length()" because "str" is null
      at ExceptionDemo.main(ExceptionDemo.java:9)
     */
}


3) StackOverflowError :
=========================
- it is unchecked. when we perform recursive method call that time we
will get this error
- StackOverFlowError <- VirtualMachineError <- Error <- Throwable
- ex.,
public class ExceptionDemo {
    public static void main(String[] args) {
       m1();
    }
    public static void m1() {
        System.out.println("Hello");
        m2();
    }
    public static void m2() {
        m1();
    }
}

4) ClassCastException :
=========================
- it is unchecked exception. When we try to typecase parent object int
child object we will get this exception
- ClassCastException <- RuntimeException <- Exception <- Throwable
-ex.,
public class ExceptionDemo {
    public static void main(String[] args) {
        Object obj = new String("Kaju");
```

```
        String str = (String )obj;      //it's valid because in object we
are storing string only
        System.out.println(str);

        String str2 = new String("Kaju");
        Object obj2 = (Object) str2;    //it's also valid because we are
converting child object to parent object
        System.out.println(obj2);

        Object obj3 = new Object();
        String str3 = (String) obj3;    //it's not valid because we are
trying to convert parent object into child object
    }
    /*
    Kaju
    Kaju
    Exception in thread "main" java.lang.ClassCastException: class
java.lang.Object cannot be cast to class java.lang.String
(java.lang.Object and java.lang.String are in module java.base of loader
'bootstrap')
          at ExceptionDemo.main(ExceptionDemo.java:17)
     */
}


5) NoClassDefFoundError :
=========================
- it is unchecked. when JVM is trying to file .class file which is not
available that time we will get this error.
- NoClassDefFoundError <- LinkageError <- Error <- Throwable

6) ExceptionInInitializerError :
=================================
- it's a unchecked. while performing static variable initialization and
static block execution if any exception raised this exception occur
- ExceptionInInitializerError <- LinkageError <- Error <- Throwable
- ex.,
public class ExceptionDemo {
    static int x = 10/0;
//    static {
//        String str = null;
//        System.out.println(str.length());
//    }
    public static void main(String[] args) {

    }
    /*
    Exception in thread "main" java.lang.ExceptionInInitializerError
    Caused by: java.lang.ArithmeticException: / by zero
          at ExceptionDemo.<clinit>(ExceptionDemo.java:7)
     */
}
```

7) IllegalArgumentException :
============================
- It's unchecked. If we are invoking method with illegal argument then we will get this exception
- IllegalArgumentException <- RuntimeException <- Exception <- Throwable
- ex.,

```
public class ExceptionDemo {
    public static void main(String[] args) {
        Thread td = new Thread();
        td.setPriority(10);
        td.setPriority(100);    // we can set priority between 1-10 only. here we are trying to set priorty more than 10 so we will get exception here
    }
   /*
   Exception in thread "main" java.lang.IllegalArgumentException
      at java.base/java.lang.Thread.setPriority(Thread.java:1138)
      at ExceptionDemo.main(ExceptionDemo.java:10)
    */
}
```

8) NumberFormatException :
============================
- it's unchecked exception. If we are trying to convert string which is not manner in number and we are trying to convert into number then we will get this exception
- NumberFormatException <- IllegalArgumentException <- RuntimeException <- Exception <- Throwable
-ex.,

```
public class ExceptionDemo {
    public static void main(String[] args) {
        String str = "Kaju";
        int res = Integer.parseInt(str);
    }
    /*
    Exception in thread "main" java.lang.NumberFormatException: For input string: "Kaju"
      at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
      at java.base/java.lang.Integer.parseInt(Integer.java:668)
      at java.base/java.lang.Integer.parseInt(Integer.java:784)
      at ExceptionDemo.main(ExceptionDemo.java:9)
     */
}
```

9) IllegalStateException :
=========================
- it's unchecked. when we are trying to call wrong time then we will get this error
- for ex., if we are calling sleeping person for waiting
- IllegalStateException <- RuntimeException <- Exception <- Throwable

- ex.,

```
public class ExceptionDemo {
    public static void main(String[] args) {
        Thread td = new Thread();
        td.start();
        td.start();
    }
    /*
    Exception in thread "main" java.lang.IllegalThreadStateException
      at java.base/java.lang.Thread.start(Thread.java:793)
      at ExceptionDemo.main(ExceptionDemo.java:10)
     */
}

public class ExceptionDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(8);
        list.add(0);

        Iterator itr = list.iterator();
        while(itr.hasNext()) {
            //Object obj = itr.next();
            itr.remove();   //here we are tryint to remove object
whithout storing or calling .next method that's why we will get exception
        }
        System.out.println(list);
    }
    /*
    Exception in thread "main" java.lang.IllegalStateException
      at java.base/java.util.ArrayList$Itr.remove(ArrayList.java:980)
      at ExceptionDemo.main(ExceptionDemo.java:18)
     */
}
```

10) AssertionError :
==========

- while debugging if assert statement failed
- AssertionError <- Error <- Throwable

Try with resources (1.7 version) :
****

```
try( BR br = new BR(new FR("input.txt"))) {
    use br based on our requirement once control reaches end of itry
block automatically
    br will be closed. we are not required to close the explicitely.
}
catch(Exception e) {
    //handling exception
}
```

1) we can take any number of resources

```
try(r1; r2; r3) {

}
```

2) all resources should be AutoClosable(I) : this interface introduced in 1.7 version and this interface include only one method
public void close() throws exception;

3) you can't reassign that resource which provided means all resources impilicitely final

```
public class ExceptionDemo {
    public static void main(String[] args) throws IOException {
        try (FileReader fr = new FileReader("input.txt")){
            fr = new FileReader("abc.txt");
        }
    }
    /*
    java: auto-closeable resource fr may not be assigned
     */
}
```

4) from 1.7 version try block without catch or finally allowed with resources.

Multi Catch Block (1.7 version) :
****
- in 1.6 version we have to do like follow :

```
try {
    //code
}
catch(AE e) {
    e.printStackTrace();
}
catch(NPE e) {
    e.printStackTrace();
}
catch(CCE e) {
    e.getMessage();
}
catch(IOE e) {
    e.getMessage();
}
```

- from 1.7 version we can do like follow to reduce the complexity of program if handling method is same
but parent child relationship exception we can't take ther for ex., AE and E

```
try {
    //code
}
catch(AE | NPE e) { //it means AE or NPE
    e.printStackTrace();
}
```

```
catch(CCE | IOE e) {
    e.getMessage();
}

- ex.,
public class ExceptionDemo {
    public static void main(String[] args) throws IOException {
        try {
            System.out.println(10/0);
            String s = null;
            System.out.println(s.length());
        }
        catch(ArithmeticException | NullPointerException e) {
            System.out.println(e.getMessage());
        }
    }
}
```