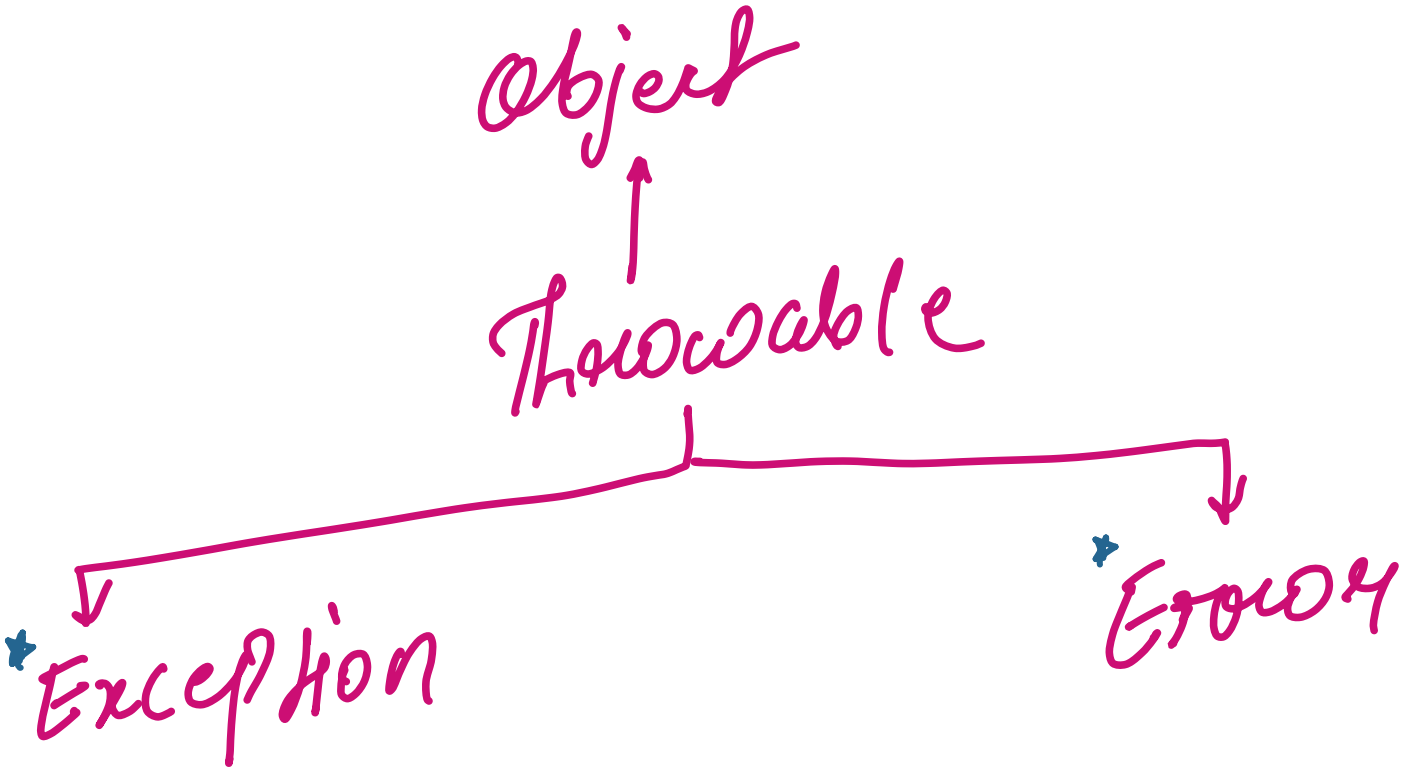
Exception

↳ Unwanted or unexception event occurs during the execution of the program at the runtime.

Object → Parent class of All the class in java

Throwable → Parent class of Exception class



Difference B/w Exception & Error

Exception

① occured by a program

② Handlable / Recoverable

③ Types
- Compile Time Exception / checked Exception
- Runtime Exception / unchecked Exception

Error

① bcoz of lack of system resources
- less RAM
- less storage

② Non Recoverable

③ Error
- Runtime

-Exception Handling Exception handling in Java is one of the powerful mechanisms to handle the runtime errors so that normal flow of the application can be maintained.

-What is exception?
An Exception is an unwanted or unexpected event which occurs during the execution of the program i.e., at run time, that disrupts the normal flow of the program.

Object is the parent class of all the classes in java.
Throwable is the parent class of Exception class.

-Difference between Exception and Error

Exception:
1. Occurred by our program.
2. Recoverable/ Hand-able.
3. Are of 2 type-
    a) Compile time Exception/Checked Exception
    b) Runtime Exception/ Unchecked Exception

Error:
1. Occurs bcoz of lack of System Resources thus Programmer cannot do anything(less RAM, Less Memory)
    -occur at the system level or the virtual machine level.
2. Not recoverable
3. Only one type —> Runtime/unchecked Exception
e.g. OutOfMemoryError

Types of Exception There are mainly two types of exceptions: checked and unchecked where error is considered as an unchecked exception. The sun microsystem says there are three types of exceptions:
1. Checked Exception 2. Unchecked Exception 3. Error

Checked Exception :

-The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc.

-Checked exceptions are checked at compile-time.

-If these exceptions are not handled/declared in the program, you will get compilation errors.

-Can Happen even if the code is written Correctly

eg: -SQL Connection can throw SQLException if wifi is lost

-reading a file and HD crash

-reading a file from PD and someone pulled the PD

-Java Check them at compile time therefore have a try catch or throw

-Java tries to pridict them, eg: reading file, Connecting over network etc.

-Java force Exception Handling for this Scenerio

Unchecked Exception :

-The classes that extend RuntimeException are known as unchecked exceptions

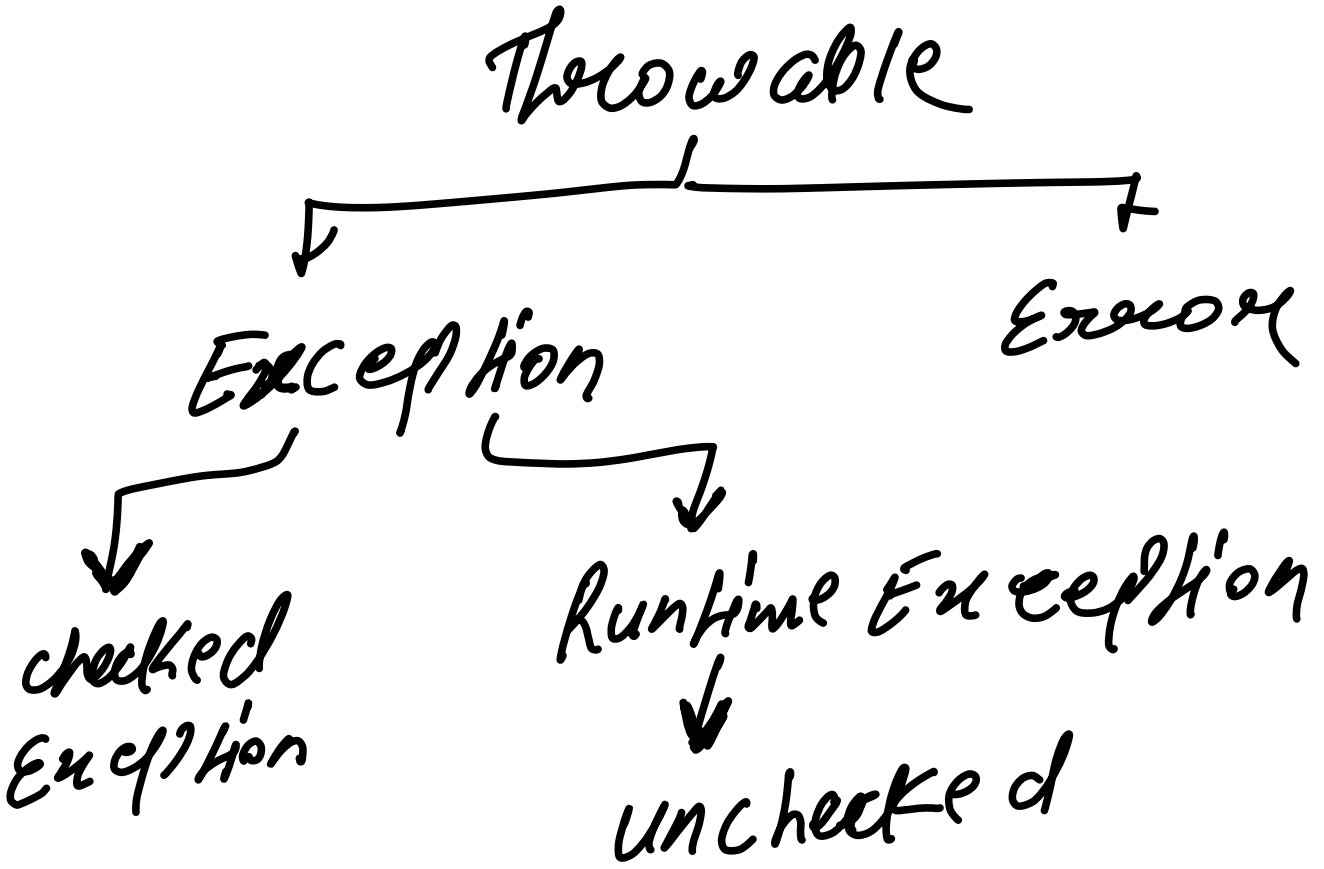
e.g. ArithmeticException, NullPointerException, , ArrayIndexOutOfBoundsException etc.

-Generally wont occur in well written program

-Unchecked exceptions are not checked at compile-time rather they are checked at runtime therefore java doesnot force you to handle them.

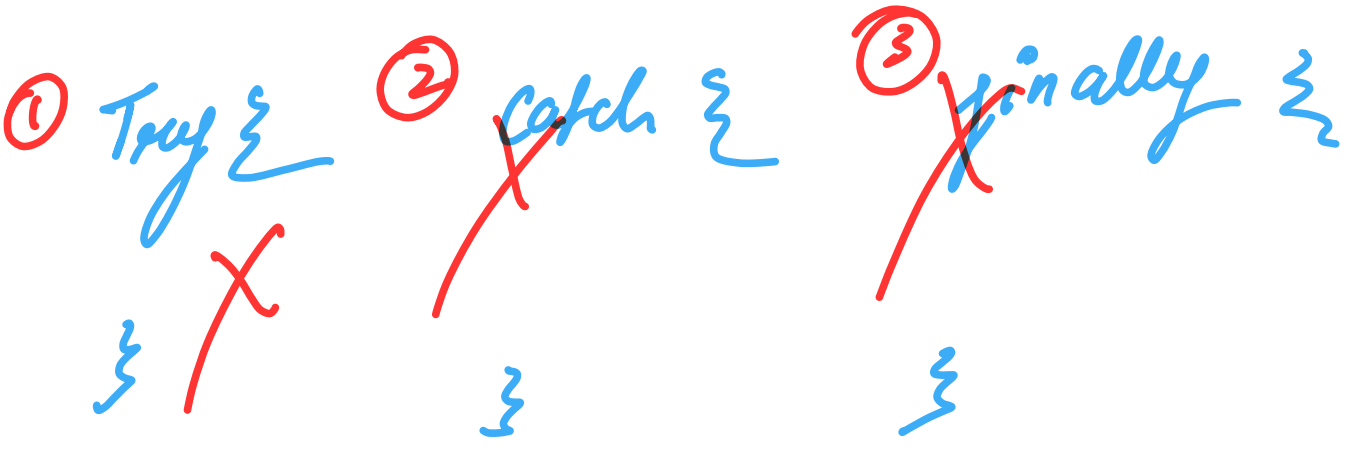
*Checked Exception directly Derived from Exception Class & Unchecked Exception are directly derived from Runtime Exception

*CE are mandatory to handle but UCE are not mandatory to handle .



Exception Handling Keywords: There are 5 keywords used in java exception handling:

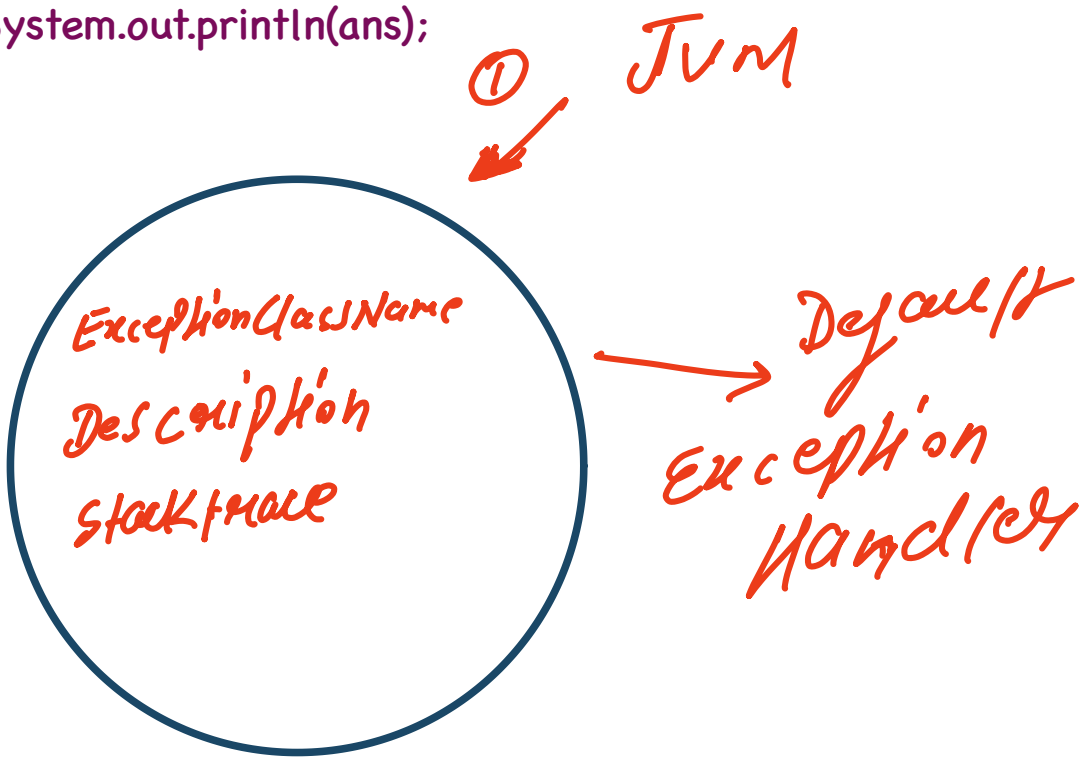
1. try
2. catch
3. finally
4. throw
5. throws



```
try {
    // Risky code
}
catch (ExceptionClassName e) {
    // Handle code
}
```

```
public class Demo1 {
    public static void main(String[] args) {
            int a = 100;
            int b = 0 ;
            int ans = a/b ;
            System.out.println(ans);

    }
}
```



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

#the Above code will compile, compiler will not able to check the Exception but this will show error in runtime

-Whenever there is a Exception, the method in which Exception Occures will create an object and that Object will store three things:

1. Exception name—> Class name of Exception
2. Description —> which type of Exception is
3. stack trace —> kis line and method me exception hai

eg: String name = null ;
System.out.println(name.length()) ;

Syntax of try & catch :

```
try{

    //riskycode

}

catch(ExceptionClassName e){

// handle code

}
```

==========

```
psvm(){

try{

int a = 100 ;
int b = 0 ;
int c = a/b ;
System.out.println(c) ;

}
catch(ArithmeticException e){            //we can also written Exception in place of ArithmeticException

System.out.println(e) ;
        or
System.out.println("You cannot divide by Zero") ;

}
```

 # Methods to print Exception Information in java(3 ways):

1. e.printStackTrace() ;
-mostly used
-print all 3 details(Exception name, Description, StackTrace)

2. System.out.println(e) or System.out.println(e.toString()) ;
-doesnot print Stack Trace

3. System.out.println(e.getMessage) ;

finally

-finally is the block that is always Executed whether Exception is handled or not

```
Syntax :
try{
//risky code
}
catch(Exception e){
//handled code
}

finally{
cleanup code
}
```

-if Exception occurs ==> try—>catch—>finally
if no Exception occurs ==> try—> finally will execute

* we can use multiple catch blocks with one try block but we can only use single finally block with one try block
* the statement present in the finally block execute even if the try block contains control transfer statement(i.e., jump statement) like return, break, continue .

#4 condition when finally block will not execute:
1. if we write System.exit() in try.
2. causing a fatal error that causes the process to abort like out of memory.
3. Exception occurs in the finally block itself and we doesnot handle it.
4. death of the thread .
5. Infinite Loop

finally Block → Always Execute Weather the Exception is Handled or Not

```
finally {
    // code
}
```

⇒

```
try {
}
catch (Exception e) {
}
finally {
}
```

try {
//
}
catch(_) {
//
}
finally {
//
}

① If No Exception occures

↳ 1st try will Execute & then finally will Execute

② If Exception occurs

↓

1st try

↓

catch

↓

finally

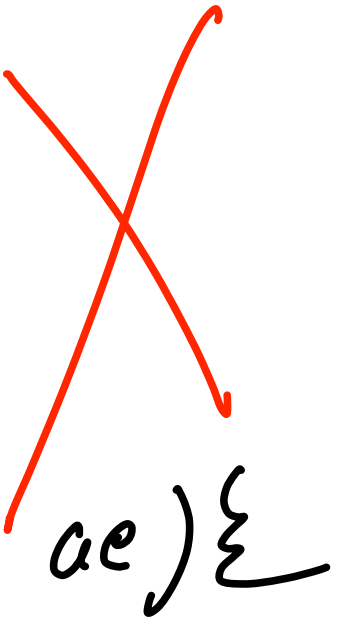
Q: When finally block will Not Execute

- If we write System.exit() inside try block
- fatal crash of the System / lack of system resource
- Thread is Dead
- If Exception occures inside the finally block itself
- Infinik loop

```
try {
    try{
    }catch {
    }
}
```

```
finally {
    try {
    }
    catch {
    }
}
```

```
try {
    //
}
catch (Exception e) {
    //
}
catch (ArithmaticException ae) {
    //
}
finally {
    //
}
```



```
try {
    //
}
catch (ArithmaticException ae) {
    //
}
catch (NPE ne) {
    //
}
catch (Exception e) {

}
```

Throw Keyword :

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

when the exception occurs the main method will create an Exception object(Exception className, Description(message), Stacketrace(location)) , and JVM will ask main() whether main has handled the Exception or not—> NO —> then JVM will terminate this method abnormally and pass this object to Default Exception Handler and then Default Exception Handler will print this object.

Syntax of throw—>

```
throw new ExceptionName("_____");
```

```
class Test{
        psvm(){
                throw new Exception
                    or
                thorw new ArithmaticException() ;
       }
}
public class Main {
    public static void main(String[] args) {
        int a = 0;
        int b = 12;


        try {
            int c = b/a;
            throw new ArithmeticException();
        }
        catch(ArithmeticException exc){
            exc.printStackTrace();


        }
    }
}
```

- Earlier main is creating the Exception object but now programmer is creating the Exception object
- this time programmer has send the Exception object to JVM instead

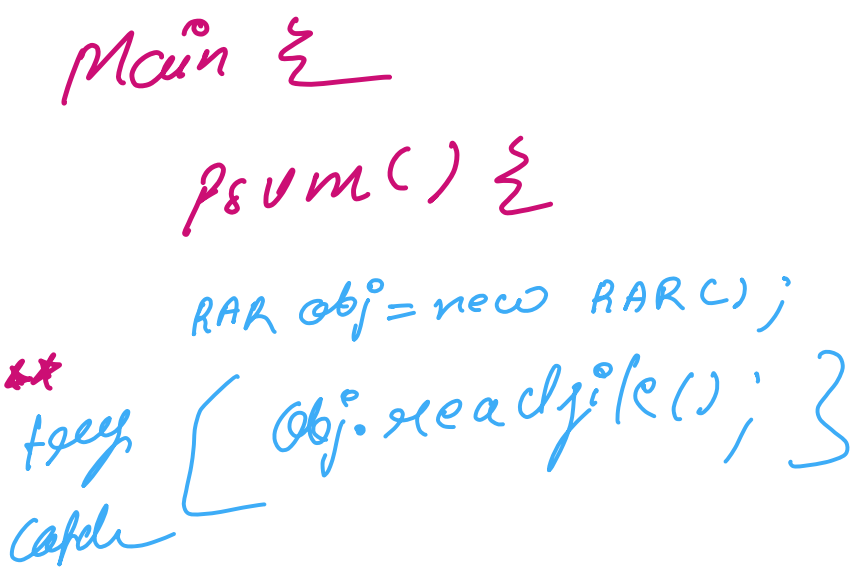
throws

- used to declare an Exception.
- It indicate the caller method that there might be an Exception so its better to use the try-catch / throws to handle that exception.

→ used with method

```
class ReadAndWrite {
    void readfile() throws FNFE {
    FileInputStream fis = new FileInputStream("-");
          - - - -
        - - - - -

    }
}
```



throws Keyword :

"throws" keyword is used to declare an Exception. It gives an indication to the caller method that there may occur an exception so it is better for the caller method to provide Exception Handling code so that normal flow can be maintained.

```
class ReadAndWrite{
    void readFile() throws FileNotFoundException{
        FileInputStream fir = new FileInputStream("d:/abc.txt") ;
                    ----------------------------
   }
}
class Test{
    psvm(){
      ReadAndWrite rw  = new ReadAndWrite() ;
          try{
              rw.readFile() ;
          }
          catch(FNFE e){
              ---------
          }
   }
}
```

now the caller method of above has to handle exception

*FileInputStream class throws "FileNotFoundException " which is a compile time Exception so we have to handle the exception and for this purpose we have to use either try catch or throws keyword

* throws keyword is always used in the case of Checked Exception not in UCE bcoz UCE is due to programmers mistake.

```
==========================================
public class Main {
    public static void main(String[] args) throws FileNotFoundException{
        ReadAndWrite raw = new ReadAndWrite() ;
        raw.readFile();
    }
}
```

— X — X — X —

throw

① used to create an Exception object mannually

② Use for runtime Exception or customized Exception

③ used within the method

④ Only Single Exception

5.The throw keyword is followed by an instance of Exception.

throws

① It is used to declare an Exception. It gives the indication to the caller method - - - - - - -

② Use for CE

③ used with the method

④ Can declare multiple Exception.

5.The throw keyword is

⑥ We Cannot write any statement after throw

⑥ No such rule

Customized Exception

⇒ checked Exception extends → Exception

⇒ unchecked Exception → RuntimeException

```
class Voting {
    psvm(_) {
        int age = 17;
        if(age<18) {
            throw new UnderAgeException();
        }
        else {
            s.o.println("you can vote");
        }
    }
}
```

```
class UnderAgeException extends Exception{
    UnderAgeException(){
        super();
    }
}
```

Customized Exception --> Checked Exception

```
public class UnderAgeException extends Exception{

    UnderAgeException(){
        super("You are under age");
    }
}
```

```
public class Voting {
    public static void main(String[] args) {
        int age = 16 ;
        try{
            if(age<18){
                throw new UnderAgeException() ;
            }
            else{
                System.out.println("You can vote");
            }
        }
        catch (UnderAgeException e){
            e.printStackTrace();

        }
    }
}
```

==========

```
public class Voting {
    public static void main(String[] args) throws
UnderAgeException{
        int age = 16 ;
        if(age<18){
            throw new UnderAgeException() ;
        }
        else{
            System.out.println("You can vote");
        }
    }
}
```

```
public class Voting {
    public static void main(String[] args) throws UnderAgeException{
        int age = 16 ;
        if(age<18){
            throw new UnderAgeException("you are under age") ;
        }
        else{
            System.out.println("You can vote");
        }
    }
}
```

Customized Exception --> Unchecked Exception

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

public class Voting {
    public static void main(String[] args){
        int age = 16 ;
        if(age<18){
            throw new UnderAgeException("you are under age") ;
        }
        else{
            System.out.println("You can vote");
        }
    }
}
```