*\*For ever thread JVM is going to create a Runtime Stack. Each and every method performed by the thread will be stored in Runtime stack after method execution is over , corresponding method will be removed from the stack. Once stack is empty the stack will be destroyed by JVM.*

***Exception working mechanism:-***

*Inside any method if exception rises that method is responsible for creating object and handing over the exception to JVM. Then JVM checks if there is any exception handling mechanism in that method, if not then JVM abnormally terminates the execution without executing rest line of code. Then JVM moves towards calling method and repeats the same process and checks if exception is handled or not in whole Runtime Stack till the main method and checks if main method has any exception handling code if not present then it will abnormally terminate main method. To handle these situation JVM maintains an assistant named as DefaultExceptionHandler present in JVM. It prints excess information on console and terminates program abnormally.*

*The method where exception occurs is responsible for handling that particular exception.*

***Exceptions:-***

*Most of the times exceptions are caused by our program and these are recoverable.*

***Eg:-*** *If programming requirement is to read data from remote file locating at some other part of world, At runtime if remote file is not available then we will get runtime exception saying* ***FileNotFoundException****. If this exception occurs we can provide local file and continue rest program normally.*

***Errors:-***

*Most of the times Errors are not caused by our program and these are due to lack of System resources. Errors are non-recoverable.*

***Eg:-*** *If* ***OutOfMemoryError*** *occurs being a programmer we can’t do anything and the program will be terminated abnormally. System admin or server admin is responsible to increase Heap memory.*

***Checked Exception:-***

* *The exception which are checked by compiler for smooth execution of program at runtime , those exceptions are called as Checked Exceptions.*
* *In our program if there is chance of Checked Exception then compulsorily we need to handle that exception by try-catch or throws keyword, or we will get compile time error.*

***Unchecked Exceptions:-***

* *The exceptions which are not checked by the compiler whether programmer is handling or not, such type of exceptions are called as Unchecked Exceptions.*

***Note:-***

* *Whether it is checked or unchecked exception , every exception occurs at Runtime only.*
* *There is no chance on any exception occurring at Compile time.*
* *All errors are Unchecked Exceptions.*
* *Runtime exception and its child classes , error and its child classes are Unchecked. Except these remaining are Checked Exceptions.*
* ***Fully Checked*** *Vs* ***Partially Checked***
* *A checked Exception is said to be Fully Checked if and only if all its Child classes are Checked.*
  + *Eg:- IOException, InterruptedException*
  + *A checked Exception is said as Partial Exception only if some of its child classes are unchecked. Eg:- Exception, Throwable*
* *Any exception which occurs outside try block then it is abnormal termination of code.*
* *Exception can occur in catch and finally block as well.*

***Customized Exception Handling*** 🡪 ***with try-catch block***

* *It Is highly recommended to handle exception. The code which may raise an exception is called Risky code.*
* *Then we have to define that code inside try block and corresponding handling code we need to define inside catch block.*

***Methods to print Exception information:-***

*1)e.printStackTrace() 🡪 Prints whole stack trace from calling methods with description.*

*2) e.toString() or e 🡪 Prints Exception class name and description.*

*3) e.getMessage() 🡪Only prints the description of message.*

***Internally DefaultExceptionHandler will use printStackTrace() to print information on Console.***

* *First catch block should always be of child then if present other child and last one whould be parent i.e Exception*
* *If there are multiple catch blocks then order of blocks is much important or else we will get compile time error saying exception has already been caught.*

***final*** *:- classes, method , variables*

* *Final is a modifier applicable for classes, method, variables.*
* *If a class is declared as final then we can’t extend that class, that is we cannot child class for that class i.e inheritance is not possible for final classes.*
* *If method is final then we cannot override that method in child class.*
* *If a variable is declared as final, then we cannot perform reassignment for that variable.*

***finally {} :-***

* *It is a block always associated with try catch block, to maintain clean up code.*
* *It will be executed always irrespective of whether exception arises or not , or exception handled or not handled.*
* *We can write try-finally block as well it is accepted*

***finalize() :-***

* *It is a method always invoked by Garbage Collector just before destroying an Object, to perform cleanup activities. Once finalize method completed immediately, GC destroys that object.*

***Difference between finally and finalize() because both do clean up code activity only?***

*🡪* ***finally*** *{} is associated with try-catch block and responsible for cleaning up try catch related code(The resources which we opened inside try will be closed inside finally block. Whereas* ***finalize()*** *is associated with Object so it is responsible for cleaning up object level cleanup activites.*

1. *In try-cath-finally Order is important.*
2. *Whenever we are writing try compulsory we should write either catch or finally otherwise we will get Compile time error.i.e try without catch,finally is invalid.*
3. *Whenever we are writing catch block compulsory try is required. Catch without try is invalid.*
4. *When writing finally block we need try . because finally without try is invalid.*
5. *Inside try 🡪 we can declare try,catch,finally blocks i.e nested try,catch,finally is allowed.*
6. *For try catch and finally block are mandatory.*

***Throw Keyword:-***

* *Sometimes we can create exception object explicitly and can handover to JVM manually, for this we have to use* ***throw*** *keyword.*
* *Hence main objective of throw keyword is to handover our created exception object to JVM manually.*
* *Best used for Customized / User defined exceptions.*
* *After throw statement , we are not allowed to write any statement directly, otherwise we will get compile time error saying “ unreachable code ”.*

***What is an Exception?***

*An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program instructions*

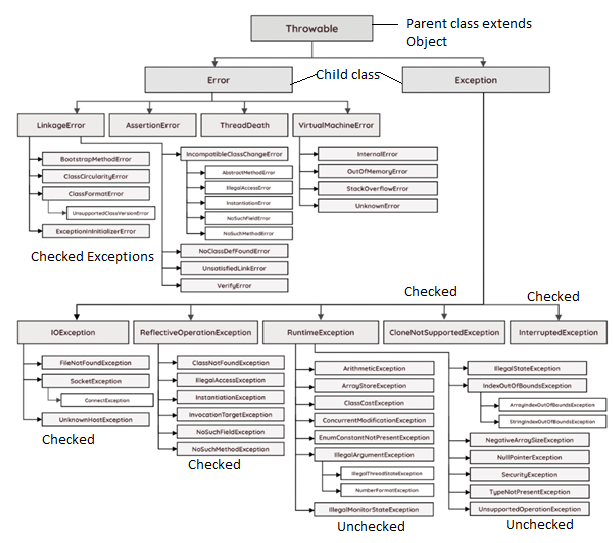
*It is* one of the powerful mechanism that handles runtime errors so that normal flow of application can be maintained.

Graceful termination of program is achieved by exception handling

***What is Exception handling in Java?***

*Exception handling is the mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException etc.*

*They are present in* ***java.lang*** *package*



***In java 99% Inbuilt Interfaces have “ able “ at the end suffix . Except Throwable*** *, because throwable is a class which is parent/super class extending grandparent as Object class.*

*Exception has two types:-*

*1) Checked Exceptions 🡪 also called* ***as Compile-Time Exception****.*

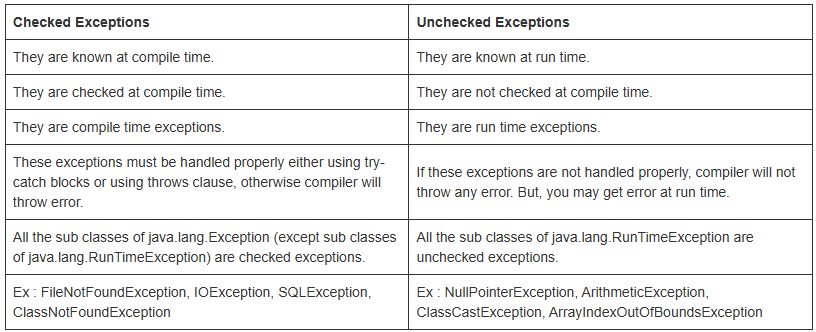
*2) Unchecked Exceptions🡪 also called* ***as Runtime Exceptions****.*

***Checked exceptions*** *are exceptions which are checked at the compile-time only.*

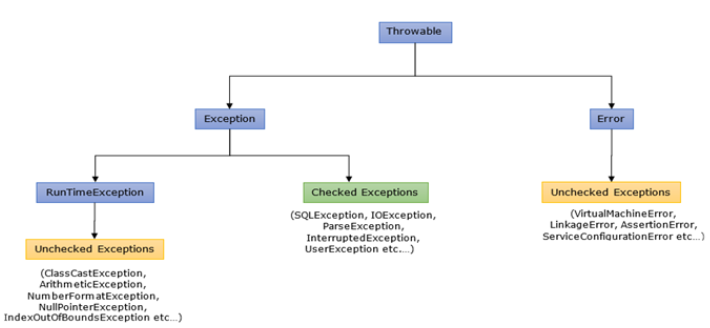
1. ***JVM takes care of Checked exceptions****.*
2. ***The classes that directly inherit the throwable class except RuntimeException and error are known as checked exception***
3. ***Checked exceptions are the exceptions which are checked during compilation itself. They are also called compile time exceptions. Compiler is aware of these exceptions and immediately throws the error wherever it sees the statements which may throw checked exceptions.***

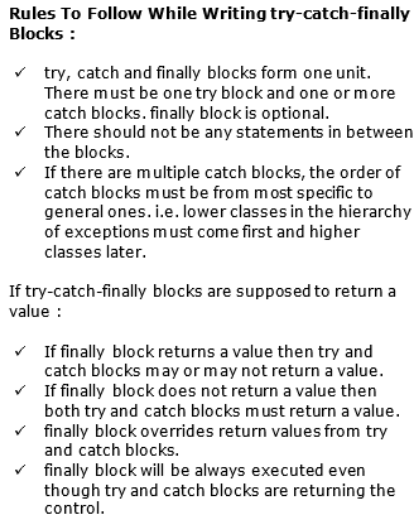
***Unchecked exceptions*** *are exceptions which are checked and occur during run-time only.*

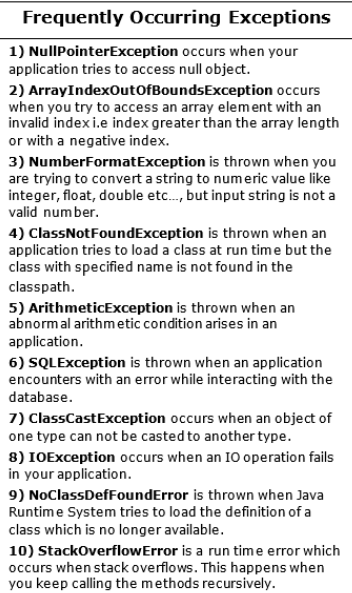
1. ***Basically they occur due to wrong user inputs and then we need to handle them to avoid disruptions of code.***
2. ***The classes that inherit Runtime Exceptions are known as Unchecked Exceptions.***
3. ***Unchecked exceptions are the exceptions which are not checked during compilation. Compiler is not aware of these exceptions. These exceptions occur only at run time. That’s why they are also called Run Time Exceptions***



***Examples of Exceptions:-***







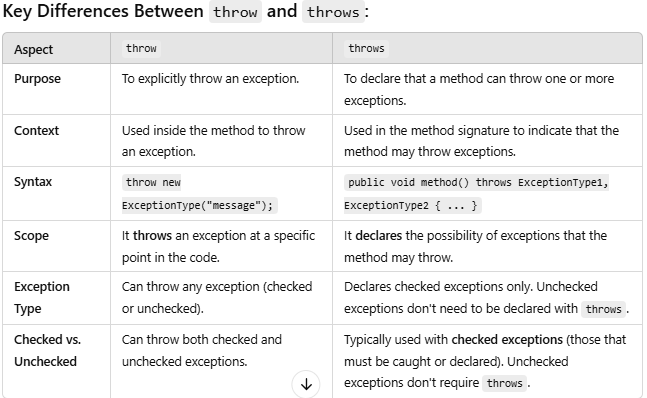
***1) try block :****This block contains statements which may throw exceptions during run time.*

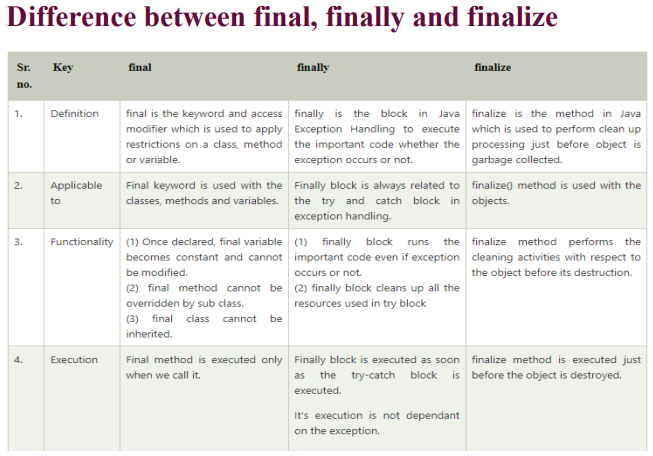
***2) catch Block :****This block handles the exceptions thrown by the try block.*

***3) finally Block :****This block is always executed whether an exception is thrown or not and thrown exception is caught or not. Hence, important code like releasing the resources are kept in this block.*

***Here’s what you can write in the finally block:***

1. ***Resource Cleanup****: Close files, database connections, or network connections.*
2. ***Releasing Locks****: If you’ve acquired any locks in the try block, you can release them in the finally block.*
3. ***Resetting Resources****: Resetting values, stopping threads, or performing any other cleanup tasks that must happen no matter what.*





***Printing Exception message ways:***

* *printStackTrace() method : It prints the name of the exception, description and complete stack trace including the line where exception occurred .*
* *getMessage() method: Mostly used, It prints the description of the exception.*
* *e.toString(): It prints name and description of the exception*

*🡪 We cannot write main method inside try-catch block.*

*🡪 We cannot write declare/define any method inside try-catch block.*

*🡪We can call methods inside try -catch blocks.*

*🡪A single try can have multiple catch Blocks*

*🡪There can be multiple try blocks sequentially /non-sequentially in a code.*

*🡪there can be nested try-catch blocks.*

*Q1) We want to write finally block but don’t want to execute it?*

* *We need to write System.exit() method at the very start of code in finally block, this will automatically throw us out of finally block without executing rest lines of code.*

*Q2) Can you run code by just writing try block without catch block and will the code compile successfully?*

***🡪 Its not a practise which we follow in our project*** *but we can achieve it by replacing catch block by ‘ finally ‘ but by doing this our exceptions wont be handled, still our code will be compiled and executed successfully.*