Exception Handling

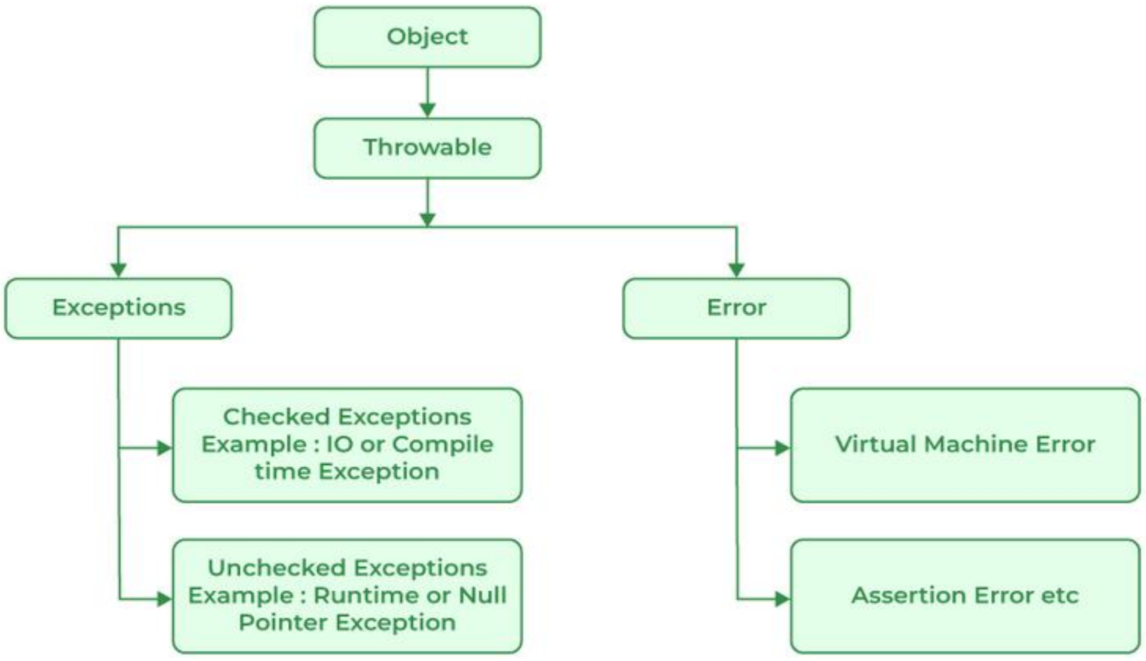
- **Exception Handling in Java**

- Dictionary Meaning: Exception is an abnormal condition.
- The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained. It is an object which is thrown at runtime.
- It handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.
- The core advantage of exception handling is to maintain the normal flow of the application

- **Difference between Exception and Error**

Sr.no	Error	Exceptions
1.	Errors primarily arise due to the lack of system resources.	Exceptions may occur during both runtime and compile time.
2.	Recovery from an error is typically not possible.	Recovery from an exception is possible
3.	All errors in Java are unchecked.	Exceptions in Java can be either checked or unchecked.
4.	The system executing the program is responsible for errors.	The program's code is responsible for exceptions

- **<u>Hierarchy of Java Exception classes</u>**
- The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error. The hierarchy of Java Exception classes is given below:


Object
Throwable
Exceptions
Error
Checked Exceptions Example : IO or Compile time Exception
Unchecked Exceptions Example : Runtime or Null Pointer Exception
Virtual Machine Error
Assertion Error etc

➢ **<u>Types of Java Exceptions</u>**

- There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error

1) Checked Exception

- The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) unchecked Exceptions

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

- Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

<u>Java Exception Keywords</u>

1) try

- Java **try** block is used to enclose the code that might throw an exception. It must be used within the method.
- If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.
- Java try block must be followed by either catch or finally block.
- <u>Syntax:</u>

try{

//code that may throw an exception }

catch(Exception_class_Name ref){}

2) Catch

- Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.
- The catch block must be used after the try block only. You can use multiple catch block with a single try block.

Java Multi-catch block

- A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
- <u>Points to remember</u>

1. At a time only one exception occurs and at a time only one catch block is executed.
2. All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Java Nested try block

- In Java, using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.
- For example, the inner try block can be used to handle ArrayIndexOutOfBoundsException while the outer try block can handle the ArithemeticException (division by zero).
- Why use nested try block
- Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

3) finally

- Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.
- Why use Java finally block?
- finally block in Java can be used to put "**cleanup**" code such as closing a file, closing connection, etc.
- The important statements to be printed can be placed in the finally block.
- Rule: For each try block there can be zero or more catch blocks, but only one finally block.
- Note: If you don't handle the exception, before terminating the program, JVM executes finally block (if any).

4) throw

- The Java throw keyword is used to throw an exception explicitly.
- We specify the **exception** object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.
- We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception.
- Syntax: **throw new** exception_class("error message");
- eg: **throw new** IOException("sorry device error");

5) throws

- The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception. So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.
- Syntax of Java throws

```
 return_type method_name() throws exception_class_name
{      //method code
 }
```

- It provides information to the caller of the method about the exception.

User-Defined Exception

```
class InvalidAgeException extends Exception
{
        InvalidAgeException(String msg)
         {
            System.out.println(msg);
         }
}
Class UserException
{
  public static void main(String args[]) throws InvalidAgeException
      {
         validateage(12);
       }
   public static void validateage(int age)   throws InvalidAgeException
{
        if(age<18)
  {
throw new InvalidAgeException("Not valid");
     }
    else
   {
System.out.println("valid age for vote");
     }
}}
Output:
Not valid
```