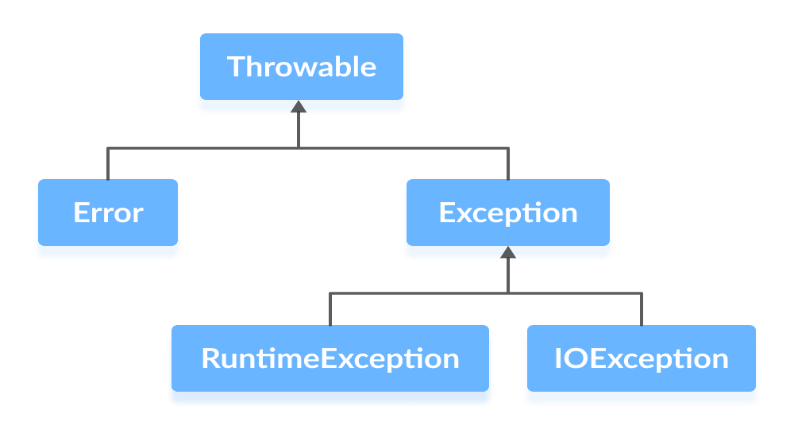
**Exception Handling**

**Java Exception hierarchy**



* Throwable class is the root class or top most interface.
* Exception is top most class.

**Errors**

* Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc.
* it is something which you can't handle. e.g JVM memory full.

**Exceptions:**

* An exception is an unexpected event that occurs during program execution. It affects the flow of the program instructions which can cause the program to terminate abnormally.
* An exception can occur for many reasons. Some of them are:

Invalid user input, Code errors

Opening an unavailable file

**Java Exception Types**

The exception hierarchy also has two branches: RuntimeException and IOException.

**1. RuntimeException**

* A runtime exception happens due to a programming error. They are also known as **unchecked exceptions.**
* These exceptions are not checked at compile-time but run-time. Some of the common runtime exceptions are:

Null pointer access (missing the initialization of a variable) - NullPointerException

Out-of-bounds array access - ArrayIndexOutOfBoundsException

Dividing a number by 0 – ArithmeticException

**2. IOException**

* An IOException is also known as a **checked exception**. They are checked by the compiler at the compile-time and the programmer is prompted to handle these exceptions.
* Some of the examples of checked exceptions are:

Trying to open a file that doesn’t exist results in FileNotFoundException

Trying to read past the end of a file

**Difference between checked & unchecked exception**

**Checked Exceptions**

* They occur at compile time.
* The compiler checks for a checked exception.
* These exceptions can be handled at the compilation time.
* It is a sub-class of the exception class.
* The JVM requires that the exception be caught and handled.
* Example of Checked exception- ‘File Not Found Exception’

**Unchecked Exceptions**

* These exceptions occur at runtime.
* The compiler doesn’t check for these kinds of exceptions.
* These kinds of exceptions can’t be caught or handled during compilation time.
* This is because the exceptions are generated due to the mistakes in the program.
* These are not a part of the ‘Exception’ class since they are runtime exceptions.
* The JVM doesn’t require the exception to be caught and handled.
* Example of Unchecked Exceptions- ‘No Such Element Exception’

**Exception Handling**

* can handle and continue the program is known as exception handling
* Here's a list of different approaches to handle exceptions in Java.

1. try...catch block
2. finally block
3. throw and throws keyword

**1. Java try...catch block**

* The try-catch block is used to handle exceptions in Java. Here's the syntax of try...catch block:

**try** {

// code

}**catch**(Exception e) {

// code }

* Here, we have placed the code that might generate an exception inside the try block. Every try block is followed by a catch block.
* When an exception occurs, it is caught by the catch block. The catch block cannot be used without the try block.
* EX:

**class** Main {

**public** **static** **void** main(String[] args) {

**try** {

// code that generate exception

**int** divideByZero = 5 / 0;

System.***out***.println("Rest of code in try block");

}

**catch** (ArithmeticException e) {

System.***out***.println("ArithmeticException => " + e.getMessage());

}

}

}

**2. Java finally block**

* In Java, the finally block is always executed no matter whether there is an exception or not.
* The finally block is optional. And, for each try block, there can be only one finally block.
* The basic syntax of finally block is:

**try** {

//code

}

**catch** (ExceptionType1 e1) {

// catch block

}

**finally** {

// finally block always executes

}

* Should be written after all the catch block.
* will be executed always regardless of exception occured or not. (in exit() it will not executed)
* close the resources(DB connection, files) in finally.
* EX:

**public** **class** FinallyExample {

**public** **static** **void** main(String [] args) {

**try** {

**int** a=10;

**int** b=0;

System.***out***.println("Value of a:" +a);

System.***out***.println("Value of b:" +b);

//System.exit(0); //finally won't get execute.

**int** c = a / b;

System.***out***.println("a / b = " + c);

} **catch** (Exception e) {

System.***out***.println("Exception Thrown:" +e);

e.printStackTrace();

//return; finally will be executed.

}

**finally** {

System.***out***.println("Finally block executed!");

}}}

**3. Java throw and throws keyword**

* The Java **throw** keyword is used to explicitly throw a single exception.
* When we throw an exception, the flow of the program moves from the try block to the catch block.
* Example:

**class** Main {

**public** **static** **void** divideByZero() {

// throw an exception

**throw** **new** ArithmeticException("Trying to divide by 0");

}

**public** **static** **void** main(String[] args) {

*divideByZero*();

}

}

* In the above example, we are explicitly throwing the ArithmeticException using the throw keyword.
* Similarly, the throws keyword is used to declare the type of exceptions that might occur within the method. It is used in the method declaration.

-------------------------------------------------------------------------------------------------------------------------------

* We use the **throws** keyword in the method declaration to declare the type of exceptions that might occur within it.
* Its syntax is:

accessModifier returnType methodName() **throws** ExceptionType1, ExceptionType2 … {

// code

}

* As you can see from the above syntax, we can use throws to declare multiple exceptions.
* Ex:

**class** Main {

**public** **static** **void** findFile() **throws** IOException {

// code that may produce IOException

File newFile=**new** File("test.txt");

FileInputStream stream=**new** FileInputStream(newFile);

}

**public** **static** **void** main(String[] args) {

**try**{

*findFile*();

} **catch**(IOException e){

System.***out***.println(e);

}}

}

* if the file test.txt does not exist, FileInputStream throws a FileNotFoundException which extends the IOException class.
* If a method does not handle exceptions, the type of exceptions that may occur within it must be specified in the throws clause so that methods further up in the call stack can handle them or specify them using throws keyword themselves.The findFile() method specifies that an IOException can be thrown. The main() method calls this method and handles the exception if it is thrown.

| **Key** | **throw** | **throws** |
| --- | --- | --- |
| Definition | Throw is a keyword which is used to throw an exception explicitly in the program inside a function or inside a block of code. | Throws is a keyword used in the method signature used to declare an exception which might get thrown by the function while executing the code. |
| Internal implementation | Internally throw is implemented as it is allowed to throw only single exception at a time i.e we cannot throw multiple exception with throw keyword. | On other hand we can declare multiple exceptions with throws keyword that could get thrown by the function where throws keyword is used. |
| Type of exception | With throw keyword we can propagate only unchecked exception i.e checked exception cannot be propagated using throw. | On other hand with throws keyword both checked and unchecked exceptions can be declared and for the propagation checked exception must use throws keyword followed by specific exception class name. |
| Syntax | Syntax wise throw keyword is followed by the instance variable. | On other hand syntax wise throws keyword is followed by exception class names. |
| Declaration | In order to use throw keyword we should know that throw keyword is used within the method. | On other hand throws keyword is used with the method signature. |

**Difference between Exception & Error:**

|  |  |
| --- | --- |
| **Exception** | **Error** |
| Exception can be recovered by using  the try-catch block. | An error cannot be recovered. |
| It can be classified into two categories  i.e. checked and unchecked. | All errors in Java are unchecked. |
| It occurs at compile time or run time. | It occurs at run time. |
| It belongs to java.lang.Exception package. | It belongs to java.lang.Error package. |
| Only checked exceptions are known to the  compiler. | Errors will not be known to the compiler. |
| It is mainly caused by the application itself. | It is mostly caused by the environment in which the application is running. |
| **CheckedExceptions:** SQLException,IOException **UncheckedExceptions:** ArrayIndexOutOfBoundException, NullPointerException, ArithmaticException | Java.lang.StackOverFlow, java.lang.OutOfMemoryError |

**Difference between ‘Final’, ‘Finally’, and ‘Finalize’**

## Final

* It is a keyword.
* It is used to apply restrictions on classes, methods and variables.
* It can’t be inherited.
* It can’t be overridden.
* Final methods can’t be inherited by any class.
* It is needed to initialize the final variable when it is being declared.
* Its value, once declared, can’t be changed or re-initialized.

## Finally

* It is a block.
* It is used to place important code in this block.
* A finally block of code always executes, whether or not an exception has occurred.

## Finalize

* It is a method.
* It is used to perform clean up processing right before the object is collected by garbage collector.

# **How can we create a custom exception in Java?**

* Sometimes it is required to develop meaningful exceptions based on the application requirements. We can create our own exceptions by extending Exception class in Java
* User-defined exceptions in Java are also known as Custom Exceptions.

**Steps to create a Custom Exception with an Example**

* CustomException class is the custom exception class this class is extending Exception class.
* Create one local variable message to store the exception message locally in the class object.
* We are passing a string argument to the constructor of the custom exception object. The constructor set the argument string to the private string message.
* toString() method is used to print out the exception message.
* We are simply throwing a CustomException using one try-catch block in the main method and observe how the string is passed while creating a custom exception. Inside the catch block, we are printing out the message.

**EX:**

1. **public** **class** CustomExceptionTest {

**public** **static** **void** main(String [] args) {

**int** num1=10,num2=20;

**try** {

**if**(num1<num2) {

**throw** **new** CustomException("Negative");

}

}**catch**(Exception e){

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

}}

}

**2)**

**class** CustomException **extends** Exception {

String message;

CustomException(String str) {

message = str;

}

**public** String toString() {

**return** ("Custom Exception Occurred : " + message);

}}

**public** **class** MainException {

**public** **static** **void** main(String args[]) {

**try** {

**throw** **new** CustomException("This is a custom message");

} **catch**(CustomException e) {

System.***out***.println(e);

}}

}

**Java Exception Handling Interview questions**

**1) What is an exception in java?**

* In java exception is an object. Exceptions are created when an abnormal situations are arised in our program. Exceptions can be created by JVM or by our application code. All Exception classes are defined in java.lang. In otherwords we can say Exception as run time error.

**2) State some situations where exceptions may arise in java?**

1) Accesing an element that does not exist in array.

2) Invalid conversion of number to string and string to number.

(NumberFormatException)

3) Invalid casting of class

(Class cast Exception)

4) Trying to create object for interface or abstract class

(Instantiation Exception)

**3) What is Exception handling in java?**

* Exception handling is a mechanism what to do when some abnormal situation arises in program. When an exception is raised in program it leads to termination of program when it is not handled properly. The
* significance of exception handling comes here in order not to terminate a program abruptly and to continue with the rest of program normally. This can be done with help of Exception handling.

**4) What is an eror in Java?**

* Error is the subclass of Throwable class in java. When errors are caused by our program we call that as
* Exception, but some times exceptions are caused due to some environment issues such as running out of
* memory. In such cases we can’t handle the exceptions. Exceptions which cannot be recovered are called
* as errors in java.
* Ex : Out of memory issues.

**5) What are advantages of Exception handling in java?**

* Separating normal code from exception handling code to avoid abnormal termination of program.
* Categorizing in to different types of Exceptions so that rather than handling all exceptions with Exception root class we can handle with specific exceptions. It is recommended to handle exceptions with specific Exception instead of handling with Exception root class.
* Call stack mechanism : If a method throws an exception and it is not handled immediately, then that exception is propagated or thrown to the caller of that method. This propogation continues till it finds an appropriate exception handler ,if it finds handler it would be handled otherwise program terminates abruptly.

**6) In how many ways we can do exception handling in java?**

* We can handle exceptions in either of the two ways :

1) By specifying try catch block where we can catch the exception.

2) Declaring a method with throws clause .

**7) List out five keywords related to Exception handling ?**

1) Try

2) Catch

3) throw

4) throws

5) finally

**8) Explain try and catch keywords in java?**

* In try block we define all exception causing code. In java try and catch forms a unit. A catch block catches
* the exception thrown by preceding try block. Catch block cannot catch an exception thrown by another try
* block. If there is no exception causing code in our program or exception is not raised in our code jvm
* ignores the try catch block.
* Syntax :

try

{

}

Catch(Exception e)

{

}

**9) Can we have try block without catch block?**

* Each try block requires atleast one catch block or finally block. A try block without catch or finally will result in compiler error. We can skip either of catch or finally block but not both.

**10) Can we have multiple catch block for a try block?**

* In some cases our code may throw more than one exception. In such case we can specify two or more catch clauses, each catch handling different type of exception. When an exception is thrown jvm checks each catch statement in order and the first one which matches the type of exception is execution and remaining catch blocks are skipped.
* Try with multiple catch blocks is highly recommended in java.
* If try with multiple catch blocks are present the order of catch blocks is very important and the order should be from child to parent.

**11) Explain importance of finally block in java?**

* Finally block is used for cleaning up of resources such as closing connections, sockets etc. if try block executes with no exceptions then finally is called after try block without executing catch block. If there is exception thrown in try block finally block executes immediately after catch block.
* If an exception is thrown,finally block will be executed even if the no catch block handles the exception.

**12) Can we have any code between try and catch blocks?**

* We shouldn’t declare any code between try and catch block. Catch block should immediately start after try block.

try{

//code

}

System.out.println(“one line of code”); // illegal

catch(Exception e){

//

}

**13) Can we have any code between try and finally blocks?**

* We shouldn’t declare any code between try and finally block. finally block should immediately start after catch block.If there is no catch block it should immediately start after try block.

try{

//code

}

System.out.println(“one line of code”); // illegal

finally{

//

}

**14) Can we catch more than one exception in single catch block?**

* From Java 7, we can catch more than one exception with single catch block. This type of handling reduces the code duplication.
* Note : When we catch more than one exception in single catch block , catch parameter is implicity final.
* We cannot assign any value to catch parameter.
* Ex : catch(ArrayIndexOutOfBoundsException || ArithmeticException e)

{

}

* In the above example e is final we cannot assign any value or modify e in catch statement.

**15) What are checked Exceptions?**

* All the subclasses of Throwable class except error,Runtime Exception and its subclasses are checked exceptions.
* Checked exception should be thrown with keyword throws or should be provided try catch block, else,the program would not compile. We do get compilation error.
* Examples :

1) IOException,

2) SQlException,

3) FileNotFoundException,

4) InvocationTargetException,

5) CloneNotSupportedException

6) ClassNotFoundException

7) InstantiationException

**16) What are unchecked exceptions in java?**

* All subclasses of RuntimeException are called unchecked exceptions. These are unchecked exceptions because compiler does not checks if a method handles or throws exceptions.
* Program compiles even if we do not catch the exception or throws the exception.
* If an exception occurs in the program,program terminates . It is difficult to handle these exceptions because there may be many places causing exceptions.
* Example :
  1. Arithmetic Exception
  2. ArrayIndexOutOfBoundsException
  3. ClassCastException
  4. IndexOutOfBoundException
  5. NullPointerException
  6. NumberFormatException
  7. StringIndexOutOfBounds
  8. UnsupportedOperationException

**17) Explain differences between checked and Unchecked exceptions in java?**

**Unchecked Exception Checked Exception**

1) All the subclasses of RuntimeException are

called unchecked exception.

All subclasses of Throwable class except

RuntimeException are called as checked exceptions

2) Unchecked exceptions need not be handled at

compile time

Checked Exceptions need to be handled at compile

time.

4) ArrayIndexOutOfBoundsException,

ClassCastException, IndexOutOfBoundException

SqlException,

FileNotFoundException,ClassNotFoundException

**69) What is default Exception handling in java?**

* When JVM detects exception causing code, it constructs a new exception handling object by including the following information.
  + Name of Exception
  + Description about the Exception
  + Location of Exception.
* After creation of object by JVM it checks whether there is exception handling code or not. If there is exception handling code then exception handles and continues the program. If there is no exception handling code JVM give the responsibility of exception handling to default handler and terminates abruptly.
* Default Exception handler displays description of exception,prints the stacktrace and location of exception and terminates the program.
* Note : The main disadvantage of this default exception handling is program terminates abruptly.

**70) Explain throw keyword in java?**

* Generally JVM throws the exception and we handle the exceptions by using try catch block. But there are situations where we have to throw userdefined exceptions or runtime exceptions.
* In such case we use throw keyword to throw exception explicitly.
  + - Syntax : throw throwableInstance;
* Throwable instance must be of type throwable or any of its subclasses.
* After the throw statement execution stops and subsequent statements are not executed. Once exception object is thrown JVM checks is there any catch block to handle the exception.
* If not then the next catchstatement till it finds the appropriate handler. If appropriate handler is not found ,then default exception handler halts the program and prints the description and location of exception.
* In general we use throw keyword for throwing userdefined or customized exception.

**71) Can we write any code after throw statement?**

* After throw statement jvm stop execution and subsequent statements are not executed. If we try to write any statement after throw we do get compile time error saying unreachable code.

**72) Explain importance of throws keyword in java?**

* Throws statement is used at the end of method signature to indicate that an exception of a given type may be thrown from the method.
* The main purpose of throws keyword is to delegate responsibility of exception handling to the caller methods, in the case of checked exception.
* In the case of unchecked exceptions, it is not required to use throws keyword.We can use throws keyword only for throwable types otherwise compile time error saying incompatible types.
* An error is unchecked , it is not required to handle by try catch or by throws.

Syntax : Class Test{

Public static void main(String args[]) throws IE

{

}}

* Note : The method should throw only checked exceptions and subclasses of checked exceptions.
* It is not recommended to specify exception superclasses in the throws class when the actual exceptions thrown in the method are instances of their subclass.

**73) Explain the importance of finally over return statement?**

* finally block is more important than return statement when both are present in a program. For example if there is any return statement present inside try or catch block , and finally block is also present first finally statement will be executed and then return statement will be considered.

**74) Explain a situation where finally block will not be executed?**

* Finally block will not be executed whenever jvm shutdowns. If we use system.exit(0) in try statement finally block if present will not be executed.

**75) Can we use catch statement for checked exceptions?**

* If there is no chance of raising an exception in our code then we can’t declare catch block for handling checked exceptions .This raises compile time error if we try to handle checked exceptions when there is no possibility of causing exception.

**76) What are user defined exceptions?**

* To create customized error messages we use userdefined exceptions. We can create user defined exceptions as checked or unchecked exceptions.
* We can create user defined exceptions that extend Exception class or subclasses of checked exceptions so that userdefined exception becomes checked.
* Userdefined exceptions can extend RuntimeException to create userdefined unchecked exceptions.
* Note : It is recommended to keep our customized exception class as unchecked,i.e we need to extend Runtime Exception class but not Excpetion class.

**77) Can we rethrow the same exception from catch handler?**

* Yes we can rethrow the same exception from our catch handler. If we want to rethrow checked exception from a catch block we need to declare that exception.

**78) Can we nested try statements in java?**

* Yes try statements can be nested. We can declare try statements inside the block of another try statement.

**79) Explain the importance of throwable class and its methods?**

* Throwable class is the root class for Exceptions. All exceptions are derived from this throwable class. The two main subclasses of Throwable are Exception and Error. The three methods defined in throwable class are :

1) void printStackTrace() :

This prints the exception information in the following format :

Name of the exception, description followed by stack trace.

2) getMessage()

This method prints only the description of Exception.

3) toString():

It prints the name and description of Exception.

**80) Explain when ClassNotFoundException will be raised ?**

* When JVM tries to load a class by its string name, and couldn’t able to find the class classNotFoundException will be thrown.
* An example for this exception is when class name is misspelled and when we try to load the class by string name hence class cannot be found which raises ClassNotFoundException.

**81) Explain when NoClassDefFoundError will be raised ?**

* This error is thrown when JVM tries to load the class but no definition for that class is found NoClassDefFoundError will occur. The class may exist at compile time but unable to find at runtime.
* This might be due to misspelled classname at command line, or classpath is not specified properly , or the classfile with byte code is no longer available