

Exception Handling

-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

Difference between Checked and unchecked Exception-

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 predict 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

```
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 .

Various Possible Combination of try catch

1. try{

}

2. catch{


}

3. finally{

}

-we can use multiple catch block with 1 try block

```
4. try{  
  
}  
catch(Exception e){  
  
}  
catch(ArithmeticException e){  
  
}
```



Exception is the parent class of all the Exception class

```
5. try{  
  
}  
catch(ArithmeticException e){  
  
}  
catch(Exception e){
```

```
}
```

```
6. try{
```

```
}
```

```
catch(ArithmeticException e){
```

```
}
```

```
catch(ArithmeticException e){    //  
ArrayIndexOutOfBoundsException
```

```
}
```

```
7. try{
```

```
    try{
```

```
    }
```

```
    catch(){
```

```
    }
```

```
}
```

```
catch(){
```

```
}
```

```
8. try{
```

```
}
```

```
catch(){
```

```
    try{
```

```
    }
```

```
    catch(){
```

```
    }
```

```
}
```

```
9. try{
```

```
}
```

```
finally{
```

```
}
```

```
catch{
```

```
}
```

```
10. try{
```

```
}  
S.o.pln("hello") ;  
catch(    ){  
  
}
```

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), Stacktrace(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
```

```
        ArithmeticException();
```

```
    }
```

```
}
```

- 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 of main method.
- we can use but we should not use throw keyword for predefined Exception instead use for Custom Exception or User Defined Exception

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.

-The throws clause in Java exceptions encompasses two scenarios:

Case1: If you effectively manage the

exception through the use of a try/catch construct, the code will execute smoothly. Case2: If you have declared the exception by specifying "throws" with the method, but not handled by try/catch, then the exception remains unhandled and will be thrown at runtime and the program will terminate.

Difference Between throw and throws

<—

—>

Customised Exception or User Defined Exception :

-agar Checked Exception ki category me lana hai to apni class ko

Exception class ko inherit karna hoga
aur agar

-apni class ko unchecked Exception
ki category me lana hai to Runtime
Exception class ki inherit karna
hoga.

Here are the steps create a custom
exception:

- Create a new class whose name should end with an Exception like `ClassNameException`. This is a convention to differentiate an exception class from regular ones.
- Make the class extends one of the exceptions which are subtypes of the `java.lang.Exception` class. Generally,

a custom exception class always extends directly from the Exception class.

- Create a constructor with a String parameter which is the detailed message of the exception. In this constructor, simply call the super constructor and pass the message. In Java, there are two types of exceptions – checked and unchecked exceptions.

```
public class UnderAgeException extends
Exception{
    UnderAgeException(){
        super("You are UnderAge") ;
    }
}
```

```
public class Main
```

```
{ //1st try  
without try catch  
public static void main(String[] args)  
{ // instead of try Catch we can  
also use throws UnderAgeException
```

```
    int age = 18 ;  
    try{  
        if(age<18){  
            throw new  
UnderAgeException() ;  
        }  
        else{  
            System.out.println("you can  
Vote");  
        }  
    }  
    catch(Exception e){  
        System.out.println(e);  
    }  
}
```

//if Custome Exception of Checked
Exception is made then it is mandatory to
use try catch or throws keyword to handle
Exception otherwise code will not compile

#Custome UnChecked Exception:

```
class underAgeException extends  
RuntimeException{  
    // rest is same as above  
}
```

* in this code code will compile without try catch but Exception is still not handled and for that we need to use try catch block

Question: Which exception needs to be specified or declared using throws clause?

Ans) Checked exception only can be declared using throws clause, because:

- unchecked Exception: Unchecked exceptions in Java are not checked at compile-time to strike a balance between code reliability and developer productivity. It's important for developers to follow good coding practices, thoroughly test their code, and handle exceptions .
- error: Beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

```
class M{
    void method() throws
IOException{
        throw new
IOException("device error");
    }
}

public class Testthrows2{
public static void main(String args[]){
    try{
        M m=new M();
        m.method();
    }
    catch(Exception e){

System.out.println("exception handled");
    }
    System.out.println("normal
flow...");
}
```

}

}