EXCEPTION HANDLING

Uday Sharma

mrudaysharma4600@gmail.com

OVERVIEW

- try-catch block
- Multiple try-catch block
- Java Nested try
- Java Final block
- Java Throw Keyword
- Java Exception propagation
- Java Throws keyword
- Java Throw vs Throws
- Final vs Finally vs Finalize
- Exception Handling with Method Overriding
- Java custom Exception

• *Exception Handling in Java:-*

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.

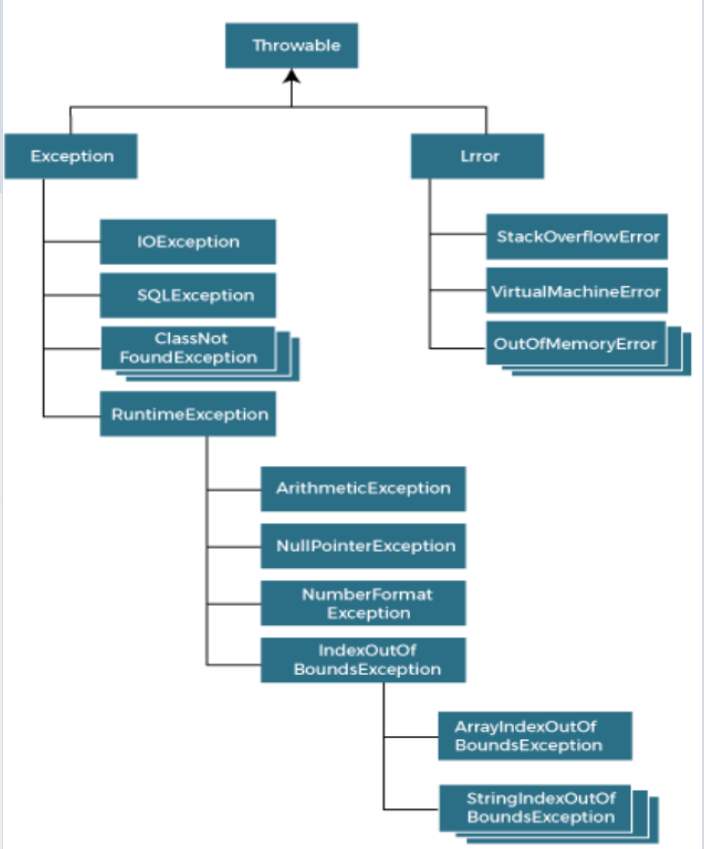
• *What is Exception Handling?*

Exception Handling is a mechanism to handle runtime errors such as **ClassNotFoundException**, **IOException**, **SQLException** , **RemoteException**, etc.

• *Advantage of Exception Handling:-*

The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario: -

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```



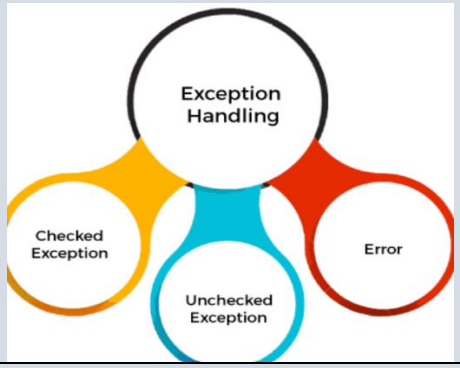
• *Hierarchy of Java Exception classes:-*

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:-

• *Types of Java Exceptions:-*

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.



• *Errors in Java:-*

There are three types of errors in java.

1) Syntax errors.

2) Logical errors.

3) Runtime errors- also called Exceptions.

- ***Difference between Checked and Unchecked Exceptions:-***

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 Exception:-

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.

- ***Java Exception Keywords:-***

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

- ***Java try-catch block:-***
- ***Java try block:-***

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.

- ***Syntax of Java try-catch:-***

```
try{
//code that may throw an exception
}catch(Exception_class_Name ref){}
```

- ***Syntax of try-finally block:-***

```
try{
//code that may throw an exception
}finally{ }
```

Code:-

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;
public class Exception_intro {
    public static void main(String[] args) {
//   ArithmeticException
   int a=100,b=0;
   try {
       int c=a/b;
       System.out.println("the value of c is :"+c);
   }catch(ArithmeticException e1){   // ArithmeticException
       System.out.println("We failed to divide. Reason: ");
       System.out.println(e1);}
//     NullPointerException
String name=null;
try {
     System.out.println("the length of name is :"+name.length());
}catch(Exception e2){   // NullPointerException
       System.out.println("We failed to print the length of the string.Reason: ");
       System.out.println(e2);}
//    ArrayIndexOutOfBoundsException
  int arr[]= {1,2,3,4,5,6,7,8,9,10};
  try {
         System.out.println("the 11th element in the array :"+arr[11]);
    }catch(ArrayIndexOutOfBoundsException e3){   // NullPointerException
           System.out.println("We failed to print the elment.Reason: ");
           System.out.println(e3);}
//           NumberFormatException
    String s="Btech-cse";
    int i=Integer.parseInt(s);
    try {
         System.out.println("converting this variable into digit will cause NumberFormatException:"+i);
    }catch(Exception e4){   // NumberFormatException
           System.out.println("We failed to run : ");
           System.out.println(e4);}
//  FileNotFoundException
    PrintWriter pw;
    try {
        pw = new PrintWriter("jtp.txt"); //may throw exception
        pw.println("saved");
    }
//providing the checked exception handler
catch (FileNotFoundException e) {
        System.out.println(e);  }
System.out.println("File saved successfully");
    }
}
```

Output:-

```
We failed to divide. Reason:
java.lang.ArithmeticException: / by zero
We failed to print the length of the string.Reason:
java.lang.NullPointerException: Cannot invoke "String.length()" because "name" is null
We failed to print the elment.Reason:
java.lang.ArrayIndexOutOfBoundsException: Index 11 out of bounds for length 10
File saved successfully
```

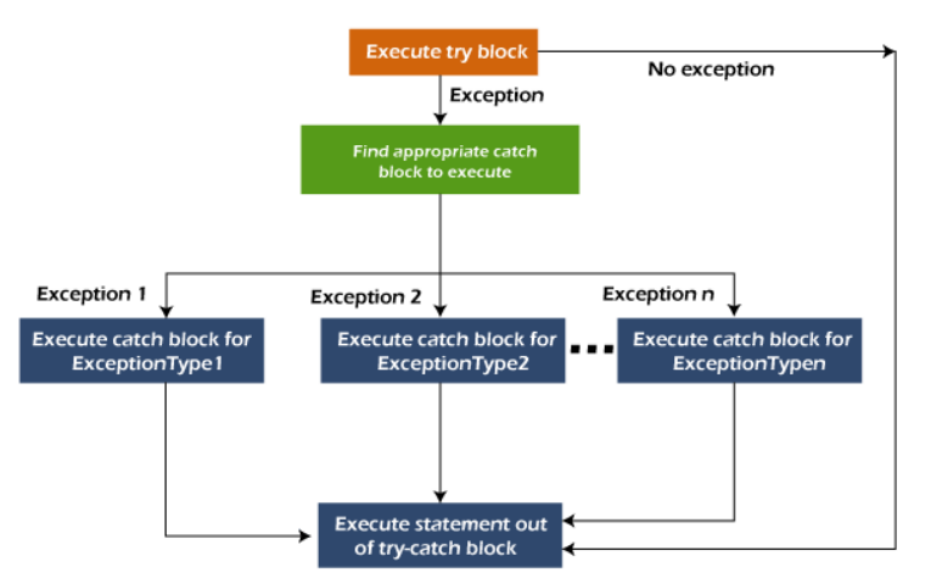
- ***Java Catch Multiple Exceptions:-***
- ***Java Multi-catch block:-***

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.

Points to remember

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

Flowchart of Multi-catch Block



Java Catch Multiple Exceptions

- ***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).

Syntax:

```
....
//main try block
try
{
   statement 1;
   statement 2;
//try catch block within another try block
   try
   {
      statement 3;
      statement 4;
//try catch block within nested try block
      try
      {
         statement 5;
```

```
        statement 6;
    }
      catch(Exception e2)
      {
//exception message
      }

   }
   catch(Exception e1)
   {
//exception message
   }
}
//catch block of parent (outer) try block
catch(Exception e3)
{
//exception message
}
```

Code:-

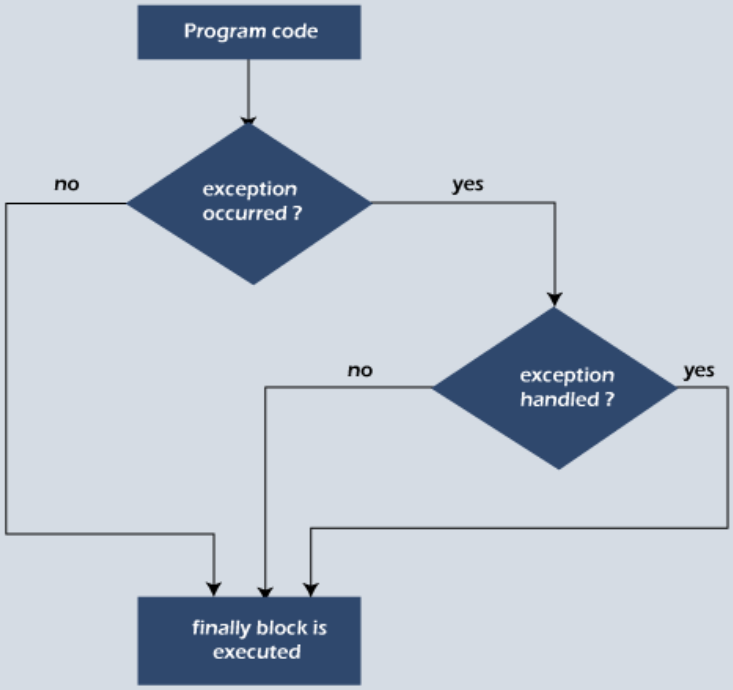
```
public class Java_Nested_Try_Block_Example {
    public static void main(String[] args) {
        try {
            try {
                System.out.println("dividing by the zero ");
                int b=10/0 ;
                System.out.println(b);
            }catch(Exception e) {System.out.println(e);}
        try {
            int a[]=new int[5];
            a[5]=90;
            System.out.println(a[5]);
        }catch(Exception e) {System.out.println(e);}

        System.out.println("other Staement ");
        } catch(Exception e) {    System.out.println("handled the exception (outer catch");}

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

Output:-

```
dividing by the zero
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
other Staement
normal flow..
```

- ***Java finally block:-***
- Java finally block is a block used to execute important code such as closing the connection, etc.
- 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.

- The finally block follows the try-catch block.
- ***Flowchart of finally block:-***



- ***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.
Usage of Java finally
Let's see the different cases where Java finally block can be used.
Code:-

```java
     // Using a finally statement
public class java_final_block_statement {
    public static void main(String[] args) {
    // using nested Try-Catch block in java
try {
        // Arithmetic Exception
try {  // Creating a try block
    System.out.println("we are in the try-1 block");
    int a=25/0;  // Getting a Arithmetic Error
    System.out.println(a);
}catch(Exception e1){  // getting an exception  in the catch block
    System.out.println("We are in the catch-1 group ");
    System.out.println(e1);  // printing an exception (e)
    }
        // ArrayIndexOfBound Exception
try {
    int a[]= {1,2,3,4,5,6,7,8,9,10};
    System.out.println("we are in the try-2 block "+a[11]);
}catch(Exception e2) {
    System.out.println("we are in the catch-2 group of Array out of bound index");
    System.out.println(e2);
}
    // NUllPointerException
try {
    String name1=null;
    System.out.println("The lenght of the String : "+name1.length());
}catch(Exception e3) {
    System.out.println("the length of the string is : ");
    System.out.println(e3);
}
        }
// Creating a finally statement , // Finally statement always occurs at last
//    its doesn't matter whether their is exception or not in the code
finally {
    System.out.println("finally block is always executed");
}
// rest of the code comment
System.out.println("rest of the code");
    }
}
```

Output:-

```
we are in the try-1 block
We are in the catch-1 group
java.lang.ArithmeticException: / by zero
we are in the catch-2 group of Array out of bound index
java.lang.ArrayIndexOutOfBoundsException: Index 11 out of bounds for length 10
the length of the string is :
java.lang.NullPointerException: Cannot invoke "String.length()" because "name1" is null
finally block is always executed
rest of the code
```

- ***Java throw Exception:-***

  In Java, exceptions allows us to write good quality codes where the errors are checked at the compile time instead of runtime and we can create custom exceptions making the code recovery and debugging easier.

- ***Java throw keyword:-***

  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.

The syntax of the Java throw keyword is given below:-
throw Instance i.e.,

- **throw new** exception_class("error message");

throw IOException.

- **throw new** IOException("sorry device error");

Where the Instance must be of type Throwable or subclass of Throwable. For example, Exception is the sub class of Throwable and the user-defined exceptions usually extend the Exception class.

- ***Java throw keyword Example:-***

Example 1: Throwing Unchecked Exception:-

In this example, we have created a method named validate() that accepts an integer as a parameter. If the age is less than 18, we are throwing the **ArithmeticException** otherwise print a message welcome to vote.

Code:-

```
import java.io.*;
public class throw_in_Exception_handling {
    public static void validate(int age) {
        if(age<18) {
            // Throwing the exception as person is not eligible for vote
            throw new ArithmeticException("person is not eligible to vote");
        }
        else {
            System.out.println("person is eligible to vote!!678");
        }
    }
    public static void main(String[] args) {
//      giving the value to the Static Function
         validate(13);
         System.out.println("the rest of the code ");
    }
}
```

Output:-

```
Exception in thread "main" java.lang.ArithmeticException: person is not eligible to vote
        at throw_in_Exception_handling.validate(throw_in_Exception_handling.java:6)
        at throw_in_Exception_handling.main(throw_in_Exception_handling.java:14)
```

Example 2: Throwing Checked Exception:-

If we throw a checked exception using throw keyword, it is must to handle the exception using catch block or the method must declare it using throws declaration.

Code:-

```
import java.io.*;
public class throw_in_Exception_handling {
 public static void Reading_a_file() throws FileNotFoundException {
     FileReader myfile=new FileReader("C:\\Users\\Arnav\\Desktop\\abc.txt");
     // passing through the bufferedReader to read the file
     BufferedReader fileinput=new BufferedReader(myfile);
     // throwing the file not found the exception
     throw new FileNotFoundException();

 }
```

```java
    public static void main(String[] args) {
        //      Running and throwing the checked Exception
        try{            // calling the method and getting the error
            Reading_a_file();
        }catch(FileNotFoundException e) {
            e.printStackTrace();
        }
        System.out.println("rest of the code");
    }
}
```

Output:-

```
java.io.FileNotFoundException: C:\Users\Arnav\Desktop\abc.txt (The system cannot find the file specified)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:216)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:111)
        at java.base/java.io.FileReader.<init>(FileReader.java:60)
        at throw_in_Exception_handling.Reading_a_file(throw_in_Exception_handling.java:12)
        at throw_in_Exception_handling.main(throw_in_Exception_handling.java:26)
rest of the code
```

Example 3: Throwing User-defined Exception:-

exception is everything else under the Throwable class.
Its to be done by extending the class into an Exception .

Code:-

```java
import java.io.*;
        // Throwing a user defined exception
class UserDefinedException extends Exception{
    public UserDefinedException(String str)
        {
            // Calling constructor of parent Exception
            super(str);
        }
    }
```

```java
public class throw_in_Exception_handling {
    public static void main(String[] args) {
        try
        {
            // throw an object of user defined exception
            throw new UserDefinedException("This is user-defined exception");
        }
        catch (UserDefinedException ude)
        {
            System.out.println("Caught the exception");
            // Print the message from MyException object
            System.out.println(ude.getMessage());
        }
        System.out.println("rest of the code");
    }
}
```

Output:-

```
Caught the exception
This is user-defined exception
rest of the code
```

- ***Java Exception Propagation:-***

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method. If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.

- Note: By default Unchecked Exceptions are forwarded in calling chain (propagated).

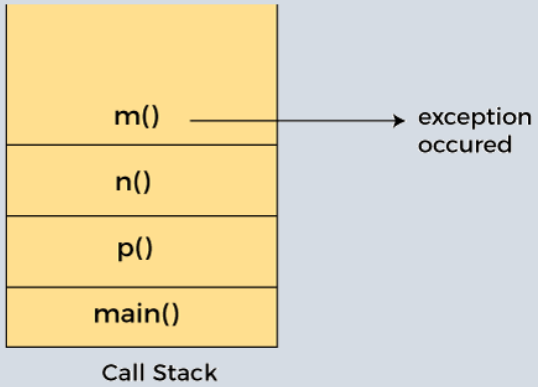
Code:-

```java
        // Showing java Exception propagation
// Means Exception Drop down from the top of the stack to the bottom until it caught
public class Java_Exception_prpagation {
    // making a function which gives the error
     public void Arithmatic_Exception() {
        // Arithmetic Exception
        int a=90,b=0;
        int data=a/b;
    }
     // ArrayIndexOfBoundException
    void ArrayIndexOfBound(){
        int a[]= {1,2,3,4,6,7,8,9};
        System.out.println(a[11]);
    }
    // NullPointerException
    void NullPointer() {
        String name1=null;
        System.out.println(name1.length());
    }
    // passing that function into the lower function of the stack
    void method2() {
        Arithmatic_Exception();
        ArrayIndexOfBound();
        NullPointer();
    }
    // passing into another function in the down of the stack
    // to caught the error
    void method3() {
//      using try-catch
        try {
            method2();
        }catch(Exception e) {  //catching the error
            System.out.println("Exception Handled-1 "+e);    }
        try {
            ArrayIndexOfBound();
        }catch(Exception e) { // catching the error
             System.out.println("Exception Handled-2 "+e);  }
        try {
            NullPointer();
        }catch(Exception e) {  // catching the error
            System.out.println("Exception Handled-3 "+e);    }
    }
    public static void main(String[] args) {
        // creating an object to access the upper function which is not static
        Java_Exception_prpagation obj=new Java_Exception_prpagation();
        // calling the method having all the error cantained method in the try-catch block
        obj.method3();
        System.out.println("The rest of the code :");
    }
}
```

Output:-

```
Exception Handled-1 java.lang.ArithmeticException: / by zero
Exception Handled-2 java.lang.ArrayIndexOutOfBoundsException: Index 11 out of bounds for length 8
Exception Handled-3 java.lang.NullPointerException: Cannot invoke "String.length()" because "name1" is null
The rest of the code :
```

Exception can be handled in any method in call stack either in the main() method, p() method, n() method or m() method.



- Note: By default, Checked Exceptions are not forwarded in calling chain (propagated).

Code:-

- ***Java throws keyword:-***

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.
Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as **NullPointerException,** it is programmers' fault that he is not checking the code before it being used.

Syntax of Java throws:-

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

- ***Which exception should be declared?***

Ans: Checked exception only, because:
unchecked exception: under our control so we can correct our code.
error: beyond our control. For example, we are unable to do anything if there occurs **VirtualMachineError** or **StackOverflowError.**

- ***Advantage of Java throws keyword:-***

Now Checked Exception can be propagated (forwarded in call stack).
It provides information to the caller of the method about the exception.

- ***Java throws Example:-***

Let's see the example of Java throws clause which describes that checked exceptions can be propagated by throws keyword.

Output:-

```
Exceeption in the catch block
This is the final block
rset of the code
```

Code:-

```java
// using the Throws Keyowrd in java
import java.io.IOException;
public class Throws_Keywprd_Exception {
    public void Checked_Exception() throws IOException{
        throw new IOException("Exception in the checked throw statment ");
    }
    public void n() throws IOException{
        Checked_Exception();
    }
    public void p() {
        try {
            n();
        }catch(Exception e) {
            System.out.println("Exceeption in the catch block");
        }
        finally {
            System.out.println("This is the final block ");
        }
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Throws_Keywprd_Exception obj=new Throws_Keywprd_Exception();
        obj.p();
        System.out.println("rset of the code ");
    }
}
```

there are two cases:

Case 1: We have caught the exception i.e. we have handled the exception using try/catch block.

Case 2: We have declared the exception i.e. specified throws keyword with the method.

- ***Case 1: Handle Exception Using try-catch block:-***

In case we handle the exception, the code will be executed fine whether exception occurs during the program or not.

Code:-

```java
// using the Throws Keyowrd in java
import java.io.IOException;
public class Throws_Keywprd_Exception {
    public void Checked_Exception() throws IOException{
        throw new IOException("Exception in the checked throw statment ");
    }
    public void n() throws IOException{
        Checked_Exception();
    }
    public void p() {
        try {
            n();
        }catch(Exception e) {
            System.out.println("Exceeption in the catch block");
        }
        finally {
            System.out.println("This is the final block ");
        }
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Throws_Keywprd_Exception obj=new Throws_Keywprd_Exception();
        obj.p();
        System.out.println("rset of the code ");
    }
}
```

```java
        // Calling the method using the object in the try-catch block
        try {
            Throws_Keywprd_Exception obj2=new Throws_Keywprd_Exception();
            obj2.Checked_Exception();
        }catch(Exception e) {  System.out.println("Exception handled");
        }
        finally{
            System.out.println("The rest of the code ");
        }
    }
}
```

Output:-

```
Exception handled
The rest of the code
```

- ***Case 2: Declare Exception:-***

1). In case we declare the exception, if exception does not occur, the code will be executed fine.
2). In case we declare the exception and the exception occurs, it will be **thrown at runtime** because throws does not handle the exception.

A) If exception does not occur :-

Code:-

```
// using the Throws Keyword in java
import java.io.IOException;
// Creating a class which contains method which is throws Exception
    class Throws_Exception_in_Java{
    // Throwing a Checked Exception method in the class
    public void Checked_Exception() throws IOException{
        throw new IOException("Exception in the checked throw statment ");
    }
    // Passing the Exception into another throws Exception method
    public void n() throws IOException{
        Checked_Exception();
    }
    // Calling that Throwed-method in the another method within the try catch block
    public void p() {
        try {
            n();
        }catch(Exception e) {
            System.out.println("Exceeption in the catch block");
        }
        finally {
            System.out.println("This is the final block ");
        }
    }
    // Declaration Exception as throws in the ,method
    public void Decalared_Exception() throws IOException{
        System.out.println("this is the Decalared Exception method Which Throws the Error in the run time");
    }
    }
public class Throws_Keywprd_Exception {
    public static void main(String[] args) throws IOException {
        // calling the method by making the object
        Throws_Exception_in_Java obj=new Throws_Exception_in_Java();
        obj.p();
        System.out.println("rest of the code-1 ");
        // Calling the method using the object in the try-catch block
        try {
            obj.Checked_Exception();
        }catch(Exception e) {  System.out.println("Exception handled");
        }
        finally{
            System.out.println("The rest of the code ");
        }
        obj.Decalared_Exception();
        System.out.println("The rest of the code");
    }
}
```

Output:-

```
Exceeption in the catch block
This is the final block
rest of the code-1
Exception handled
The rest of the code
this is the Decalared Exception method Which Throws the Error in the run time
The rest of the code
```

B) If exception occurs:-

Code:-

```
// If Exception does not occurs because we take the Throw Exception Method without contain throw keyword
        obj.Decalared_Exception();
// If Exception occurs because we take the Throw Exception Method with contain throw keyword
        obj.Checked_Exception();
        System.out.println("The rest of the code");
```

Output:-

```
this is the Decalared Exception method Which Throws the Error in the run time
java.io.IOException: Exception in the checked throw statment
        at Throws_Exception_in_Java.Checked_Exception(Throws_Keywprd_Exception.java:7)
        at Throws_Keywprd_Exception.main(Throws_Keywprd_Exception.java:46)
```

- ***Difference between throw and throws in Java:-***

1). The throw and throws is the concept of exception handling where the throw keyword throw the exception explicitly from a method or a block of code whereas the throws keyword is used in signature of the method.

2). There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

S.No	Basis of Differences	throw	throws
1.	Definition	Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
2.	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.	
3.	Syntax	The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
4.	Declaration	throw is used within the method.	throws is used with the method signature.
5.	Internal implementation	We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

- ***Java Throw and Throws-Keywords Examples:-***

Code:-

```
import java.util.*;
class Throw_Throws_Keywords {
    // creating a method which uses the throw keyoword
    public void even_number(int a) {
        if(a%2==0) {
            System.out.println("The number is divisble by 2 this is even number ");
        else {
        throw new ArithmeticException("this is not even number ");   }
       }
    // creating a method which uses the throws keyword
    public int number_divided_by_0(int b) throws ArithmeticException{
        int c=b/0;
        return c;   }
}
```

```
public class Java_Throw_and_Throws_keyword {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner sc=new Scanner(System.in);
        Throw_Throws_Keywords obj1=new Throw_Throws_Keywords();
        // checking the throw-Keyword exception
        System.out.println("Enter the number whether its even or odd :");
        int n=sc.nextInt();
        obj1.even_number(n);
        // checking the throws-keyword exception
        System.out.println("Enter the number which dived by the zero :");
        n=sc.nextInt();
        try {
            System.out.println(obj1.number_divided_by_0(n));
        }catch(Exception e) {
            System.out.println("this cannot be divided by the zero : "+e);   }
        finally {
            System.out.println("the rest of the code : ");   }
    }
}
```

Output of throw-keyword:-

```
Enter the number whether its even or odd :
15
Exception in thread "main" java.lang.ArithmeticException: this is not even number
        at Throw_Throws_Keywords.even_number(Java_Throw_and_Throws_keyword.java:8)
        at Java_Throw_and_Throws_keyword.main(Java_Throw_and_Throws_keyword.java:23)
```

Output of throws-keyword:-

```
Enter the number which dived by the zero :
45
this cannot be divided by the zero : java.lang.ArithmeticException: / by zero
the rest of the code :
```

- ***Using both throw and throws-keyword:-***

code:-

```
public void Checked_Exception() throws IOException{
    throw new IOException("Exception in the checked throw statment ");
}
```

- ***Difference between final, finally and finalize***

The final, finally, and finalize are keywords in Java that are used in exception handling. Each of these keywords has a different functionality. The basic difference between final, finally and finalize is that

the **final** is an access modifier, **finally** is the block in Exception Handling and **finalize** is the method of object class.

Along with this, there are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

S.No	Key	final	finally	finalize
1.	Definition	final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.
2.	Applicable to	Final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	finalize() method is used with the objects.
3.	Functionality	(1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.
4.	Execution	Final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. It's execution is not dependent on the exception.	finalize method is executed just before the object is destroyed.

1. *Java final Example:-*

Code:-

```
        // Final-Keyword
// used to access modifier which is used to apply restriction
//After Declaring final-Keyword its become constant and cannot
// be modified ,cannot be overridden or inherited in other sub class
public class Java_final {
// Assigning the variable as final
    final int age=18;
    void display() {
// it cannot be changed because it is final keyword
        age=34; // throw an error
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Java_final obj1 =new Java_final();
        obj1.display(); // calling that function
    }
}
```

Output:-

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The final field Java_final.age cannot be assigned

        at Java_final.display(Java_final.java:8)
        at Java_final.main(Java_final.java:13)
```

2. Java finally keyword:-

Code:- we already seen the finally Keyword is used after the try-catch Block , finally block statement is always executes regardless of exception occurred or not in the try-catch block .

- ***Java finalize Example:-***

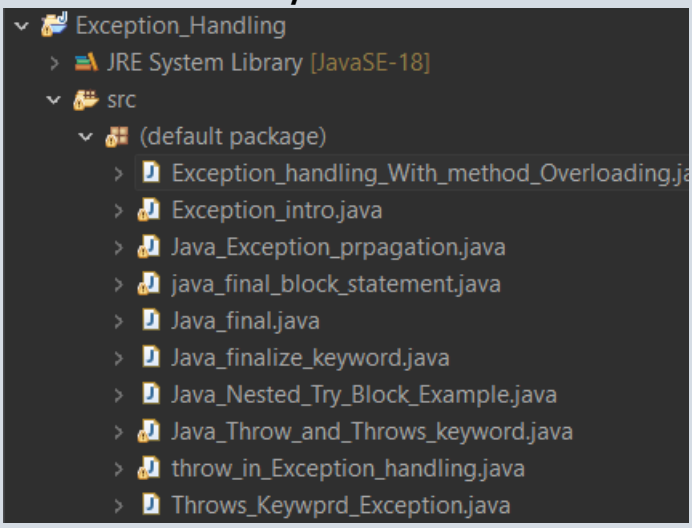
Code:-

```
     //finalize keyword
public class Java_finalize_keyword {
    public static void main(String[] args) {
        Java_finalize_keyword obj=new Java_finalize_keyword();
        // printing The hashCode
        System.out.println("the hash code is : "+obj.hashCode());
        obj=null;
        System.gc();
        System.out.println("the end of the garbage collection : ");
    }
    // defining the finalize method
    protected void finalize() {
        System.out.println("calling the finalize() method :");
    }
}
```

Output:-

```
the hash code is : 640070680
the end of the garbage collection :
calling the finalize method :
```

The Topics Covered



- Go check out my ***LinkedIn profile*** for more notes and other resources content

@Uday Sharma

mrudaysharma4600@gmail.com

https://www.linkedin.com/in/uday-sharma-602b33267