EXCEPTION HANDLING

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 in Java?
an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is Exception Handling?

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

EXCEPTION HANDLING IN JAVA

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:

EXCEPTION HANDLING IN JAVA

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

EXCEPTION HANDLING IN JAVA

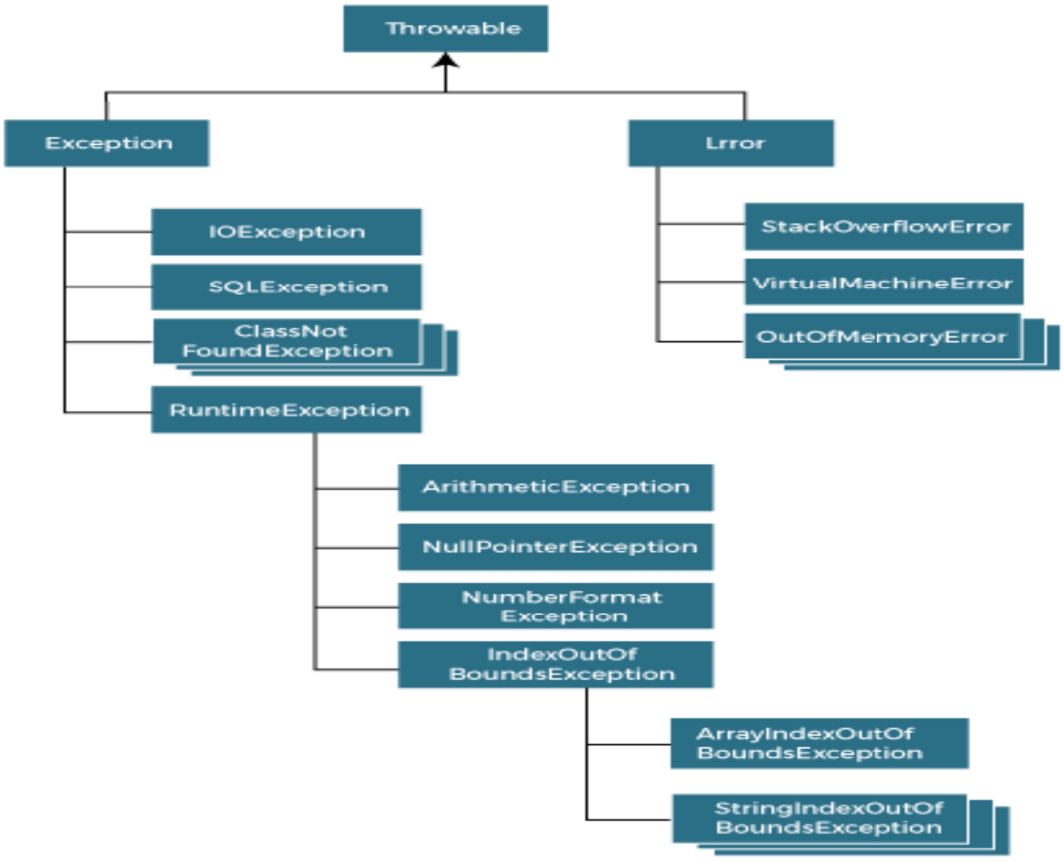
Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed.

However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

EXCEPTION HANDLING IN JAVA

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:



EXCEPTION HANDLING IN JAVA

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception.

Checked Exception
Unchecked Exception

EXCEPTION HANDLING IN JAVA

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.

EXCEPTION HANDLING IN JAVA

Java Exception Keywords
1. 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.

2. 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.

EXCEPTION HANDLING IN JAVA

3. finally
The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

4. throw
The "throw" keyword is used to throw an exception.

5. 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.

EXCEPTION HANDLING IN JAVA

Common Scenarios of Java Exceptions
There are given some scenarios where unchecked exceptions may occur. They are as follows:

1) A scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

EXCEPTION HANDLING IN JAVA

2) A scenario where ArrayIndexOutOfBoundsException occurs

When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException. Consider the following statements.

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

EXCEPTION HANDLING IN JAVA

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.

EXCEPTION HANDLING IN JAVA

Syntax of Java try-catch

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

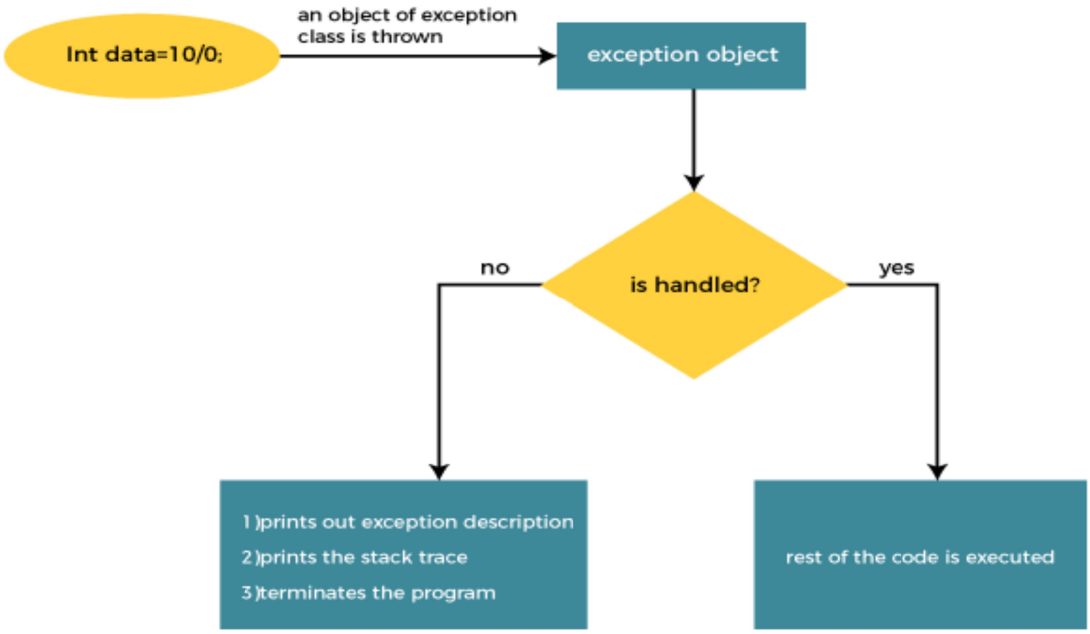
EXCEPTION HANDLING IN JAVA

Java catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter.
The catch block must be used after the try block only. You can use multiple catch block with a single try block.

EXCEPTION HANDLING IN JAVA

Internal Working of Java try-catch block



EXCEPTION HANDLING IN JAVA

Problem without exception handling

```
public class TryCatchExample1 {

    public static void main(String[] args) {

        int data=50/0; //may throw exception

        System.out.println("rest of the code");

    }

}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

EXCEPTION HANDLING IN JAVA

Solution by exception handling

```
public class TryCatchExample2 {

    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception
        }
            //handling the exception
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }
        System.out.println("rest of the code");
    }

}
```

```
java.lang.ArithmeticException: / by zero
rest of the code
```

EXCEPTION HANDLING IN JAVA

print a custom message on exception

```
public class TryCatchExample5 {

    public static void main(String[] args) {
        try
        {
        int data=50/0; //may throw exception
        }
             // handling the exception
        catch(Exception e)
        {
                  // displaying the custom message
            System.out.println("Can't divided by zero");
        }
    }
}
```

```
Can't divided by zero
```

EXCEPTION HANDLING IN JAVA

resolve the exception in a catch block

```
public class TryCatchExample6 {

    public static void main(String[] args) {
        int i=50;
        int j=0;
        int data;
        try
        {
        data=i/j; //may throw exception
        }
            // handling the exception
        catch(Exception e)
        {
             // resolving the exception in catch block
            System.out.println(i/(j+2));
        }
    }
}
```

```
25
```

EXCEPTION HANDLING IN JAVA

handle another unchecked exception

```
public class TryCatchExample9 {

    public static void main(String[] args) {
        try
        {
        int arr[]= {1,3,5,7};
        System.out.println(arr[10]); //may throw exception
        }
            // handling the array exception
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println(e);
        }
        System.out.println("rest of the code");
    }
}
```

```
java.lang.ArrayIndexOutOfBoundsException: 10
rest of the code
```

EXCEPTION HANDLING IN JAVA

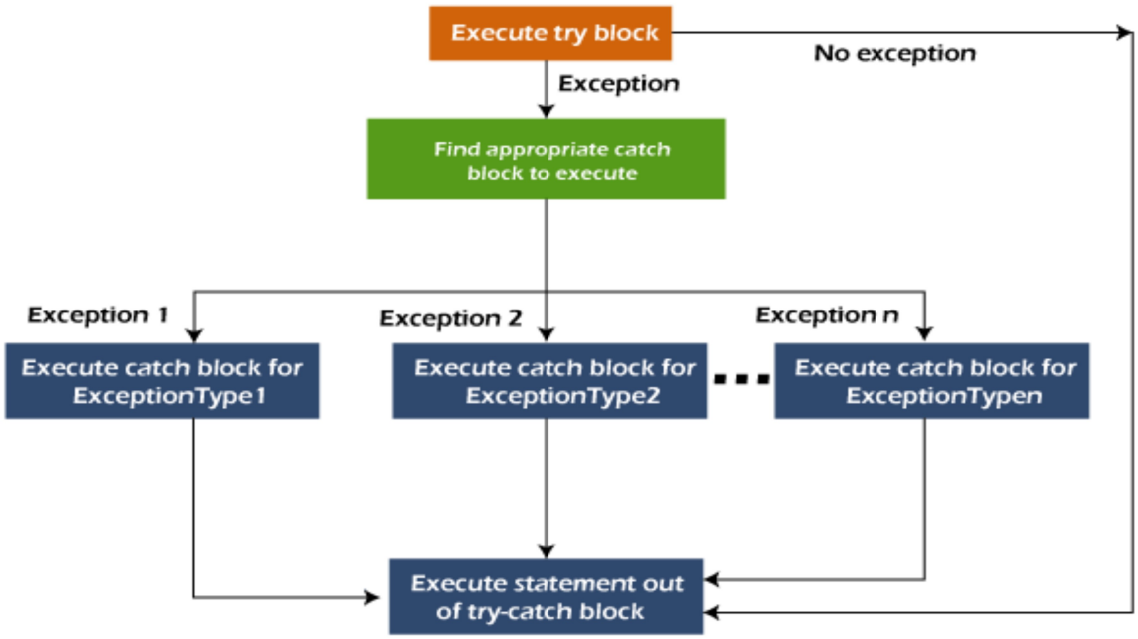
Java Catch Multiple Exceptions

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.

EXCEPTION HANDLING IN JAVA

Flowchart of Multi-catch Block



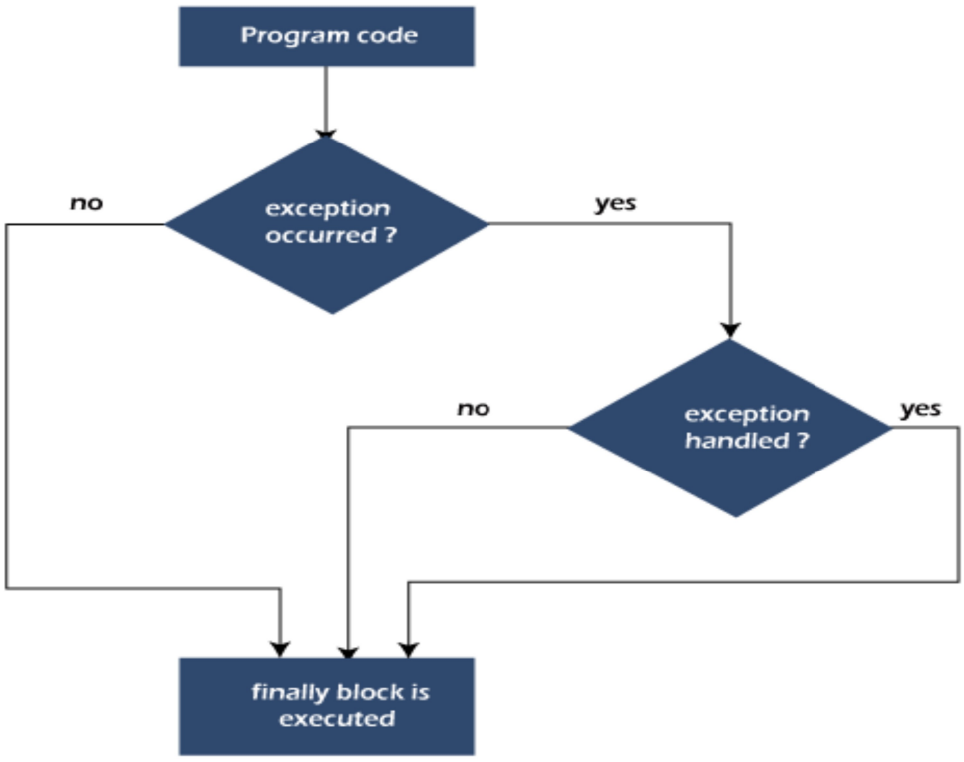
EXCEPTION HANDLING IN JAVA

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.

EXCEPTION HANDLING IN JAVA



CASE 1: WHEN AN EXCEPTION DOES NOT OCCUR

```
class TestFinallyBlock {
  public static void main(String args[]){
  try{
//below code do not throw any exception
   int data=25/5;
   System.out.println(data);
  }
//catch won't be executed
  catch(NullPointerException e){
System.out.println(e);
}
//executed regardless of exception occurred or not
 finally {
System.out.println("finally block is always executed");
}

System.out.println("rest of the code...");
  }
}
```

```
5
finally block is always executed
rest of the code...
```

CASE 2: WHEN AN EXCEPTION OCCURR BUT NOT HANDLED BY THE CATCH BLOCK

```
public class TestFinallyBlock1{
    public static void main(String args[]){

    try {
            System.out.println("Inside the try block");
           //below code throws divide by zero exception
            int data=25/0;
           System.out.println(data);
      }
    //cannot handle Arithmetic type exception
    //can only accept Null Pointer type exception
    catch(NullPointerException e){
      System.out.println(e);
    }
   //executes regardless of exception occured or not
    finally {
      System.out.println("finally block is always executed");
    }

    System.out.println("rest of the code...");
    }
}
```

```
Inside the try block
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at TestFinallyBlock1.main(TestFinallyBlock1.java:9)
```

EXCEPTION HANDLING IN JAVA

throw Exception

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.

We can throw either checked or unchecked exceptions in Java by throw keyword.

EXCEPTION HANDLING IN JAVA

We can also define our own set of conditions and throw an exception explicitly using throw keyword. For example, we can throw ArithmeticException if we divide a number by another number. Here, we just need to set the condition and throw exception using throw keyword.

The syntax of the Java throw keyword is given below.

```
throw new exception_class("error message");
```

Example

```
throw new IOException("device error");
```

EXCEPTION HANDLING IN JAVA

Throwing Unchecked Exception

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.

EXCEPTION HANDLING IN JAVA

```
public class TestThrow1 {
    //function to check if person is eligible to vote or not
    public static void validate(int age) {
        if(age<18) {
            //throw Arithmetic exception if not eligible to vote
            throw new ArithmeticException("Person is not eligible to vote");
        }
        else {
            System.out.println("Person is eligible to vote!!");
        }
    }
    //main method
    public static void main(String args[]){
        //calling the function
        validate(13);
        System.out.println("rest of the code...");
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: Person is not eligible to vote
        at TestThrow1.validate(TestThrow1.java:6)
        at TestThrow1.main(TestThrow1.java:15)
```

EXCEPTION HANDLING IN JAVA

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.

EXCEPTION HANDLING IN JAVA

```
import java.io.*;
public class TestThrow2 {

    //function to check if person is eligible to vote or not
    public static void method() throws FileNotFoundException {

        FileReader file = new FileReader("C:\\Users\\Vivek\\Desktop\\abc.txt");
        BufferedReader fileInput = new BufferedReader(file);
        throw new FileNotFoundException();
    }
    //main method
    public static void main(String args[]){
        try
        {
            method();
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        System.out.println("rest of the code...");
  }
}
```

EXCEPTION HANDLING IN JAVA

throws keyword

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
}

EXCEPTION HANDLING IN JAVA

```
import java.io.*;
class M{
 void method()throws IOException{
   throw new IOException("device error");
  }
}
class Testthrows3{
   public static void main(String args[])throws IOException{//declare exception
     M m=new M();
     m.method();
  }
}
```