**Exception Handling in Java**

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

Here, we will learn about Java exceptions, its type and the difference between checked and unchecked exceptions.

**What is Exception in Java**

**Dictionary Meaning:** Exception is an abnormal condition.

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.

**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 use exception handling. Let's take a scenario:

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

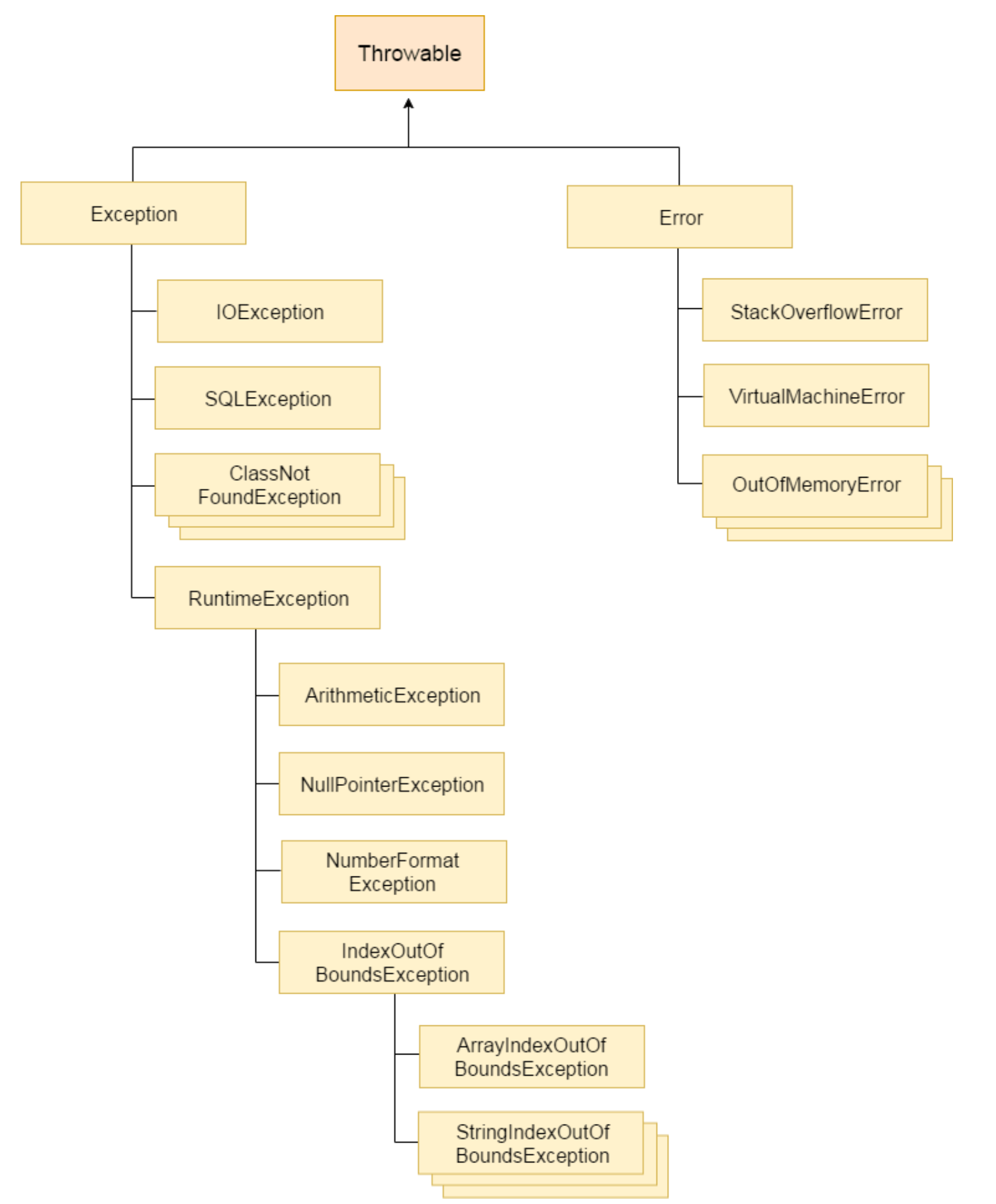
statement 9;

statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

**Hierarchy of Java Exception classes**

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



**Types of Java Exceptions**

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

**Difference between Checked and Unchecked Exceptions**

1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

**Java Exception Handling Example**

public class JavaExceptionExample{

public static void main(String args[]){

try{

//code that may raise exception

int data=100/0;

}catch(ArithmeticException e){System.out.println(e);}

//rest code of the program

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

}

}

**Common Scenarios of Java Exceptions**

int a=50/0;//ArithmeticException

String s=null;

System.out.println(s.length());//NullPointerException

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

**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 of try block, the rest of the block code will not execute. So, it is recommended not to keeping 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{

}

**Java catch block**

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.

**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");

}

}

**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 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**

At a time only one exception occurs and at a time only one catch block is executed.

All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Let's see a simple example of java multi-catch block.

public class MultipleCatchBlock1 {

public static void main(String[] args) {

try{

int a[]=new int[5];

a[5]=30/0;

}

catch(ArithmeticException e)

{

System.out.println("Arithmetic Exception occurs");

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("ArrayIndexOutOfBounds Exception occurs");

}

catch(Exception e)

{

System.out.println("Parent Exception occurs");

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

**Java Nested try block**

The try block within a try block is known as nested try block in java.

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.

class Excep6{

public static void main(String args[]){

try{

try{

System.out.println("going to divide");

int b =39/0;

}catch(ArithmeticException e){System.out.println(e);}

try{

int a[]=new int[5];

a[5]=4;

}catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

System.out.println("other statement”);

}catch(Exception e){System.out.println("handeled");}

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

}

}

**Java finally block**

Java finally block is a block that is used to execute important code such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

**Why use java finally**

Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

class TestFinallyBlock{

public static void main(String args[]){

try{

int data=25/5;

System.out.println(data);

}

catch(NullPointerException e){System.out.println(e);}

finally{System.out.println("finally block is always executed");}

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

}

}

**Java throw keyword**

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

**The syntax of java throw keyword is given below.**

throw exception;

**java throw keyword example**

public class TestThrow1{

static void validate(int age){

if(age<18)

throw new ArithmeticException("not valid");

else

System.out.println("welcome to vote");

}

public static void main(String args[]){

validate(13);

System.out.println("rest of the 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 normal flow 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 performing check up before the code 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 your control so correct your code.

error: beyond your control e.g. you 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.

class M{

void method()throws IOException{

throw new IOException("device error");

}

}

class Testthrows4{

public static void main(String args[])throws IOException{//declare exception

M m=new M();

m.method();

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

}

}

**Java finalize example**

class FinalizeExample{

public void finalize(){System.out.println("finalize called");}

public static void main(String[] args){

FinalizeExample f1=new FinalizeExample();

FinalizeExample f2=new FinalizeExample();

f1=null;

f2=null;

System.gc();

}}

**Difference between final, finally and finalize**

