Chapter 5

Exception Handling

Introduction

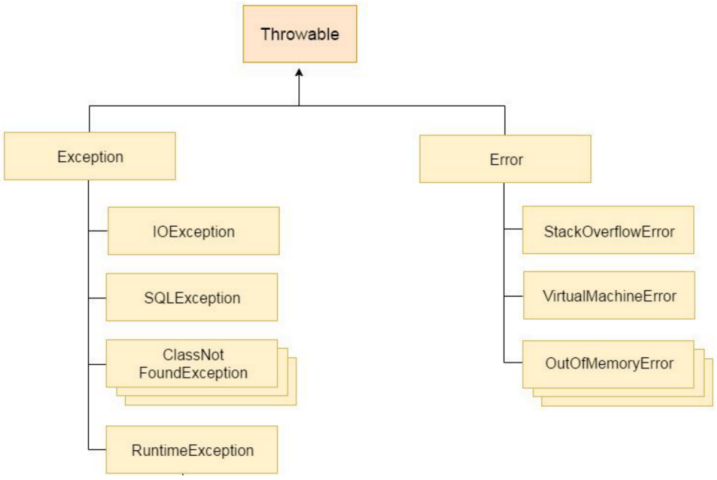
The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained. In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

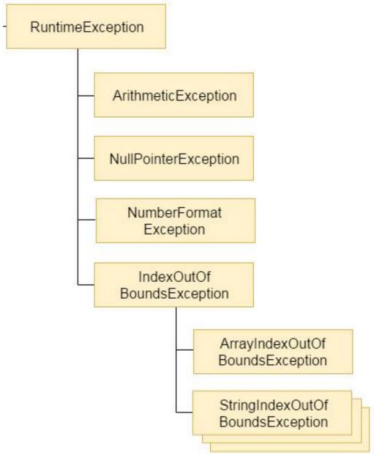
Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

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

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.
- Exception class is inherited by classes like IOException,SQLException, ClassNotFoundException and RuntimeException.
- Error class is inherited by Errors like StackOverFlowError, Virtual MachineError and OutofMemoryError class.





Types of Java Exceptions

- **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.
- **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.
- **Error:** Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
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 important 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 doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

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 program
         System.out.println("Rest Code of Program");
   }
}
```

Output:

```
java.lang.ArithmeticException: / by zero
Rest Code of Program
```

• **A scenario where ArithmeticException occurs**

— If we divide any number by zero, there occurs an ArithmeticException.
- int a=50/0; //ArithmeticException

• **A scenario where NullPointerException occurs**

— If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.
- String s=null;
- System.out.println(s.length()); //NullPointerException

• **A scenario where NumberFormatException occurs**

— The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.
- String s="abc";
- int i=Integer.parseInt(s);//NumberFormatException

• **A scenario where ArrayIndexOutOfBoundsException occurs**

— If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:
- int a[]=new int[5];
- a[10]=50; //ArrayIndexOutOfBoundsException

Syntax of Java try-catch

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

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.

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

Java multiple catch example

```
public class MultipleCatch Block{
public static void main(String[] args) {
try{
int a[]=new int[5];
System.out.println(a[10]);
}
catch(ArithmeticException e)
{
System.out.println("ArithmeticException 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 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.

•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...");

}

}
```

Output:

```
5

finally block is always executed

rest of the code...
```

Java throw keyword

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

• We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

Syntax:
throw exception;
Example:
throw new IOException("sorry device error");

```
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...");
}
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:not valid
```

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

```
class Throwsdemo {
public static void main(String args[]) {
try {
System.out.println(divide(2, 1));
}
catch (IOException e) {
System.out.println(e);
}
}
public static int divide(int a,int b) throws IOException
{
return a/b;
}
}
```

Java Custom Exception

• If we are creating our own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

• Custom Exception is created by extending the class from Exception class and then overriding the to String() method.

```
class customDemo {
public static void main(String args[]) {
try {
int a=1;
if(a==1)
{
throw new CustomException();
}
}
catch (CustomException ex) {
System.out.println(ex);
}
}
}
```

```
class CustomException extends Exception{ public String toString()
{
return "This is custom error";
}
}
```