Exception Handling

Exceptional handling: The block of code that react to a specific type of exception. If the exception is for error that the program can recover from, the program can resume executing after the execution handler has executed .Java provides us facility to create our own exceptions which are basically derived classes of Exception.

Exception handling is one of the most important features of java programming that allows us to handle the runtime errors caused by exceptions. In this guide, we will learn what is an exception, types of it, exception classes and how to handle exceptions in java with examples.

What is an exception?

An Exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs program execution gets terminated. In such cases we get a system generated error message. The good thing about exceptions is that they can be handled in Java. By handling the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message, which may not be understandable to a user.

Why an exception occurs?

There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc.

Exception Handling

If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user. For example look at the system generated exception below:
An exception generated by the system is given below

```
Exception in thread "main" java.lang.ArithmeticException: / by zero at
ExceptionDemo.main(ExceptionDemo.java:5)
ExceptionDemo : The class name
main : The method name
ExceptionDemo.java : The filename
java:5 : Line number
```

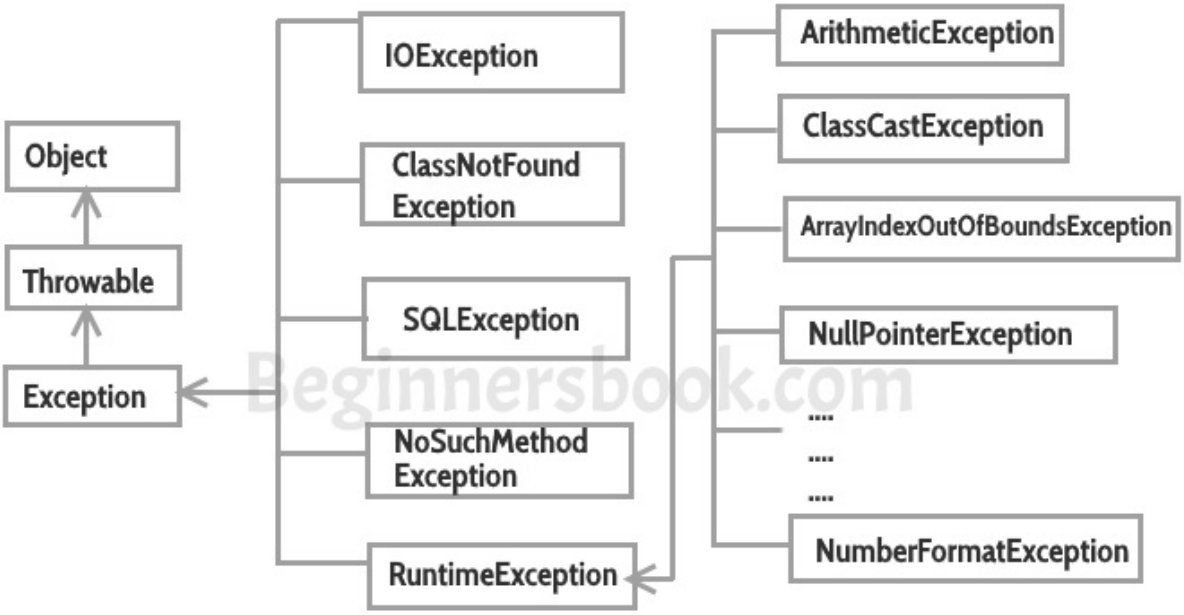
This message is not user friendly so a user will not be able to understand what went wrong. In order to let them know the reason in simple language, we handle exceptions. We handle such conditions and then prints a user friendly warning message to user, which lets them correct the error as most of the time exception occurs due to bad data provided by user.

Difference between error and exception

Errors indicate that something severe enough has gone wrong, the application should crash rather than try to handle the error.

Exceptions are events that occurs in the code. A programmer can handle such conditions and take necessary corrective actions. Few examples:

NullPointerException – When you try to use a reference that points to null.
ArithmeticException – When bad data is provided by user, for example, when you try to divide a number by zero this exception occurs because dividing a number by zero is undefined.
ArrayIndexOutOfBoundsException – When you try to access the elements of an array out of its bounds, for example array size is 5 (which means it has five elements) and you are trying to access the 10th element.



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

All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error. For example, SQLException, IOException, ClassNotFoundException etc.

2) Unchecked Exception

Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

3) Error

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

Java Exception Keywords

Keyword	Description
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.

Implementations of keywords like try, catches, finally, throw & throws.

1.try: The try block contains set of statements where an exception can occur.

```
try
{
   // statement(s) that might cause exception
}
```

2.catch : Catch block is used to handle the uncertain condition of try block. A try block is always followed by a catch block, which handles the exception that occurs in associated try block.

```
catch
{
  // statement(s) that handle an exception
  // examples, closing a connection, closing
  // file, exiting the process after writing
  // details to a log file.
}
```

3.throw: Throw keyword is used to transfer control from try block to catch block.

4.throws: Throws keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

Throw vs Throws in java

Throws clause is used to declare an exception, which means it works similar to the try-catch block. On the other hand **throw** keyword is used to throw an exception explicitly.If we see syntax wise

than **throw** is followed by an instance of Exception class and **throws** is followed by exception class names.
For example:

```
throw new ArithmeticException("Arithmetic Exception");
```

and

```
throws ArithmeticException;
```

Throw keyword is used in the method body to throw an exception, while throws is used in method signature to declare the exceptions that can occur in the statements present in the method.

5.**finally**: It is executed after catch block. We basically use it to put some common code when there are multiple catch blocks.
A **finally block** contains all the crucial statements that must be executed whether exception occurs or not. The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.

Syntax of Finally block

```
try {
   //Statements that may cause an exception
}
catch {
  //Handling exception
}
finally {
  //Statements to be executed
}
```

Advantage of exception handling

Exception handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly.
By handling we make sure that all the statements execute and the flow of program doesn't break.

Advantage 1: Separating Error-Handling Code from "Regular" Code
Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program. In traditional programming, error detection, reporting, and handling often lead to confusing spaghetti code.

Advantage 2: Propagating Errors Up the Call Stack
A second advantage of exceptions is the ability to propagate error reporting up the call stack of methods. Suppose that the readFile method is the fourth method in a series of nested method calls made by the main program

Advantage 3: Grouping and Differentiating Error Types
Because all exceptions thrown within a program are objects, grouping or categorizing of exceptions is a natural outcome of the class hierarchy. An example of a group of related exception classes in the Java platform are those defined in java.io: IOException and its descendants. IOException is the most general and represents any type of error that can occur when performing I/O. Its descendants represent more specific errors.

Importance of exception handling in practical implementation of live projects.

Exception handling is used when the frequency of occurance of an exception cannot be predicted.

Real world examples:

1. you provide a web form for users to fill in and submit.but incase there are a lot of conditions to be handled and the conditions keeps changing periodically,you use exception handling to simplify the process
2. database connectivity uses exception handling(why???) this is because the reason for database connectivity failure cannot be predicted and handled as it can be caused by many variables such as power failure, unreachable server,failure at client front/back end and so on.
3. internet communication
4. arithmetic exceptions such as division by zero and so on.
5. operating systems use exception handling to resolve deadlocks,recover from crash and so forth.